SNIFFER

PROJECTE FINAL DE CARRERA

FACULTAT DE TELECOMUNICACIONS
DE BARCELONA

Títol: SNIFFER

Volum: 1/1

Alumne: Joaquim Orra Serra

Director/Ponent: Isaac Gelado Fernández
Departament: Arquitectura de Computadors

Data: 28 de Juny del 2010

DADES DEL PROJECTE

Títol: SNIFFER

Volum: 1/1
Alumne: Joaquim Orra Serra

Director/Ponent: Isaac Gelado Fernández
Departament: Arquitectura de Computadors
Data: 28 de Juny del 2010

MEMBRES DEL TRIBUNAL

President: Carlos Villavieja Prados

Vocal: Gregorio Vázquez Grau

Secretari: Isaac Gelado Fernández

QUALIFICACIÓ

Qualificació numèrica:

Qualificació descriptiva:

Data:

ÍNDEX

ÍNDEX DE FIGURES

1 Introducció

Durant els set mesos que he estat realitzant el Projecte Final de Carrera moltes persones m'han preguntat quin projecte estava desenvolupant. Jo responia, un Sniffer. Aquesta resposta no satisfeia, per tant, vaig pensar en una altre resposta. Aquesta va ser, un capturador de tràfic d'internet. En veure que aquesta resposta tampoc acabava de satisfer, i que és un concepte difícil d'entendre pels qui son aliens a aquest món, vaig pensar en quelcom metafòric conegut per tothom.

La resposta va ser explicar el què fa la policia quan punxa una línia telefònica. Aquesta resposta va ser la definitiva.

1.1 La memòria

La memòria del Projecte Final de Carrera consta de diferents apartats.

En el capítol dos s'expliquen els diferents tipus d'sniffers que existeixen i, les diferents tecnologies de distribució i web que existeixen. En el capítol tres s'ha estudiat els diferents requisits que hauria de complir l'aplicació. El capítol quatre parla de la metodologia que he seguit junt amb una planificació econòmica i temporal. El cinquè i el sisè capítols serien els més densos que s'explica el disseny del Projecte i la seva implementació.

Un cop acabada l'aplicació cal fer una seria de probes per verificar que es compleixen els requisits establerts en el capítol tres. Aquestes probes i la seva descripció la trobem en el capítol set.

L'anàlisi econòmica, és a dir, el cost que té el projecte, la trobem en el penúltim apartat.

L'últim capítol son les conclusions que s'han extret de la realització del projecte i possibles funcionalitats noves per incorporar en un futur.

1.2 El projecte

El projecte ha tingut dues parts força diferenciades.

La primera va ser documentar-me sobre que és i com treballen els sniffers, instal·lar, fer funcionar, entendre l'aplicació d'escriptori sobre la qual vaig desenvolupar el meu projecte. Aquesta part va ser complicada degut a la meva inexperiència amb corba, amb sniffers i amb aplicacions d'escriptori.

La segona va ser el desenvolupament d'una aplicació web que es comunicava amb el servidor de captures de l'aplicació original. Aquesta segona va ser forca complicada ja que l'aplicació tenia que comunicar-se amb tres elements imprescindibles; el servidor de captures ja comentat i dos servidors d'aplicacions amb les seves màquines virtuals corresponents.

Aquestes dues parts han estat assentades cronològicament una després de l' altre ja que primer era necessari investigar i aprofundir en el què hi havia fet i després crear-ne un versió millor.

Gràcies al projecte hem passat d'un sniffer que era una aplicació d'escriptori a tenir una aplicació web que integra un sniffer. Veurem quins són els canvis que s'han realitzat a l'aplicació original per adaptar-la i també com s'ha dissenyat l'entorn web per poder albergar un servidor de captures desenvolupat amb Corba i C++.

Aquest és un projecte molt interessant pels qui vulguin endinsar-se en el món dels sniffers.

1.3 Història i precedents

La història del projecte va comen car el dia que vaig anar a parlar amb l'Isaac Gelado. Jo volia un projecte de J2EE ja que és el llenguatge de programació que utilitzo a la feina. Aquell dia em va proposar fer un projecte i em va dir que em penses si el volia portar a terme.

Vaig deixar passar l'estiu i el setembre del 2008 vaig tornar al seu despatx. Em va dir que el projecte que m'havia comentat ja estava assignat. Tot i això n'hem va proposar un altre de molt interessant. El projecte tractava d'agafar una aplicació d'escriptori i convertir-la en una aplicació web. A més a més l'aplicació era un Sniffer. La idea em va agradar tant que m'hi vaig tirar de cap.

1.4 Utilitat

Sniffer bé de l'anglès “sniff” que vol dir ensumar. És, essencialment, un programa que captura les trames de la xarxa i que generalment s'utilitza per gestionar xarxes.

És comú, degut a la topologia de xarxa, tenir un medi de transmissió compartit per diverses màquines i dispositius. Per tant un ordinador, pot capturar trames que no estan destinades a ell i d'aquesta manera gestionar un xarxa.

Permet veure i analitzar la informació real que viatge per la xarxa. Per aquest motiu els sniffers han estat utilitzats per molts hackers per capturar contrasenyes i noms

d'usuari que viatgen per la xarxa.

Permet analitzar errades a la xarxa i veure per exemple perquè un ordinador A no pot establir comunicació amb un ordinador B.

Així doncs, veiem que un sniffer és una eina molt potent i eficaç si es vol entendre com funciona una xarxa.

Els sniffers tenen altres utilitats com poden ser detectar problemes de connectivitat a una xarxa o per depurar aplicacions de xarxa. Permet estudiar les xarxes descobrint punts crítics com poden ésser els colls d'ampolla i les vulnerabilitats de la xarxa.

Els sniffers també són útils per detectar intrusos a la xarxa o per investigar les possibles hackers d'una xarxa. Molts sniffers dedicats a investigar hackers permeten la creació de registres de xarxa de manera que els hackers puguin saber que són investigats.

1.5 Objectius

El principal objectiu del meu projecte final de carrera és realitzar una aplicació web eficient i usable utilitzant el servidor de captures que utilitzava l'aplicació d'escriptori que em va proporcionar l'Isaac Gelado utilitzant tecnologies “open source”.

El primer objectiu és que l'aplicació mostri captures per pantalla. Dins d'aquest apartat podríem definir objectius menys importants però no per això menys necessaris. Guardar les captures, veure els diferents sniffers que es troben a la xarxa, poder filtrar captures per ip, port i protocol, reconfigurar el filtre de captures són objectius que s'hauran d'anar assolint un cop s'hagi dut a terme el primer objectiu esmentat anteriorment.

Un dels objectius més importants és que l'aplicació sigui eficient i ràpida. Que el codi que la forma sigui robust i les tecnologies utilitzades també. Caldrà, per tant, raonar molt bé alhora de prendre decisions de disseny de l'aplicació.

L'aplicació haurà de ser vistosa i entenedora per tant els estils i les decisions estructurals també hauran d'ésser meditades amb deteniment.

2 Estat de l'art

2.1 Introducció als sniffers

Començarem aquest apartat definint què és un sniffer. Un sniffer, bàsicament, és un programa que captura les trames de xarxa. Un sniffer pot tenir diferents utilitats com poden ser analitzar i detectar errors o punts dèbils a la xarxa. També es pot utilitzar per a la enginyeria inversa de protocols. Els sniffers també es poden utilitzar per a capturar contrasenyes, escoltar chats, interceptar correus electrònics, etc.

2.2 La llibreria libpcap

La majoria d'sniffers que corren en sistemes Unix tenen com a llibreria base “libpcap”.

Libpcap és una llibreria “open source” , la qual, ofereix una api per poder realitzar captures de tràfic. Inclou un analitzador sintàctic per a cadenes de filtrat cosa què la fa relativament fàcil d'utilitzar. Es troba disponible per a C i C++. Existeixen però “wrappers” que són fines capes que transformen les interfícies de les llibreries perquè siguin entendibles per altres llenguatges. Per pcap existeixen wrappers per a Java i .NET.

Cal dir que per a windows també existeix una versió “winpcap”. Aquestes llibreries contenen mètodes de baix nivell, que permeten fer captures de tràfic a aplicacions web o d'escriptori. Tot i això cal no pensar que libpcap i winpcap són la mateixa llibreria, mentre libpcap accedeix a un mòdul dins del kernel, winpcap no ho fa ja que els sistemes windows no implementen aquest mòdul. El filtrat el realitzen dins de la mateixa llibreria amb la qual cosa la fa menys eficient.

Per a utilitzar aquesta llibreria dins qualsevol aplicació primer de tot cal buscar un dispositiu per començar la captura, extreure la direcció de xarxa i la seva mascara, definir el filtre de les captures, compilar-lo i relacionar-lo amb la captura i finalment començar la captura. La captura serà en mode promiscu, és a dir, que capturarà tots els paquets de xarxa encara que no passin per la màquina a on es troba intallat l'sniffer. Per tant, en aquest cas, serà necessari que l'usuari executi l'sniffer com a Administrador(super usuari).

Cal dir que libpcap també té mètodes que detecten possibles errors que puguin fer inviable les captures.

2.3 Història de les captures de tràfic

Les captures de tràfic es van començar a utilitzar sobretot en sistemes UNIX ja que es va comprovar que eren molt eficaces per monitoritzar i detectar errades i punts dèbils

a les xarxes. Per a què aquesta tècnica sigui eficaç cal que es pugui realitzar en qualsevol punt de la xarxa, especialment en zones on la xarxa pugui ser vulnerable.

Degut a la importància d'aquesta tècnica és lògic pensar que el kernel ofereixi facilitats a les aplicacions, a nivell d'usuari, per a poder accedir a aquestes dades.

El 1980 va sortir "ICMU/Standford Packet Filter (CSPF)" que oferia grans avantatges. Tot i això amb la progressiva migració cap a sistemes RISC, aquest filtre dissenyat per a màquines basades en piles de memòria, va decaure enormement. Aquestes noves arquitectures feien un ús exhaustiu del recursos.

El 1992 apareix BSD "Packet Filter" el qual constitueix una nova i millor implementació de captura de paquets en el kernel. Aquest sistema esta dissenyat especialment per màquines basades en registres. A més a més presenta un nou esquema de filtrat que el fa molt millor que els seus antecessors. De fet, avui en dia, una versió d'aquest és la més utilitzada i està present dins el kernel de molts sistemes operatius.

Els sistemes operatius basats en microkernels també es poden realitzar captures. Tot i que no es pot implantar dins el kernel un mòdul, ja que tampoc existeix una entrada/sortida, es pot implementar un sistema apart al servei TCP/IP en execució. De totes maneres no hi ha cap treball sobre aquest tema pel fet que quasi bé no hi ha sistemes operatius d'aquesta mena.

EL "BSD Packet sniffer" pot ésser utilitzat directament o a través d'una api. Actualment la única api que existeix per a sistemes UNIX és libpcap.

2.4 Captures de tràfic

En aquest capítol explicaré breument els diferents sniffers que existeixen. Tots ells permeten examinar dades d'una xarxa viva i permeten solucionar problemes en xarxes de comunicació.

Wireshark

Abans es coneixia amb el nom d' "Ethereal". És semblant a **tcpdump**, però inclou

una interfície gràfica amb la possibilitat de generar gràfiques de les captures, moltes opcions d'organització i un filtrat d'informació. És capaç de detectar problemes a la xarxa.

Els aspectes més importants són:

- Sostingut sota la llicència *GPL* (General Public License)
- Treballa amb mode promiscu i amb mode no promiscu.
- Pot llegir totes les dades de la xarxa i guardar-les en un arxiu.
- Basat en la llibreria pcap.
- Interfície molt flexible.
- Gran capacitat de filtrat.
- Admet format d'arxius tcpdump.
- S'executa en més de 20 plataformes.
- Compatible amb més de 480 protocols.
- Pot llegir arxius de captures de més de 20 productes.

Per capturar paquets normalment es necessiten permisos d'execució especials. Per aquest motiu Wireshark s'executa amb permisos de Superusuari.

En la següent figura (Figura 1) es pot veure un exemple dels gràfics que genera.

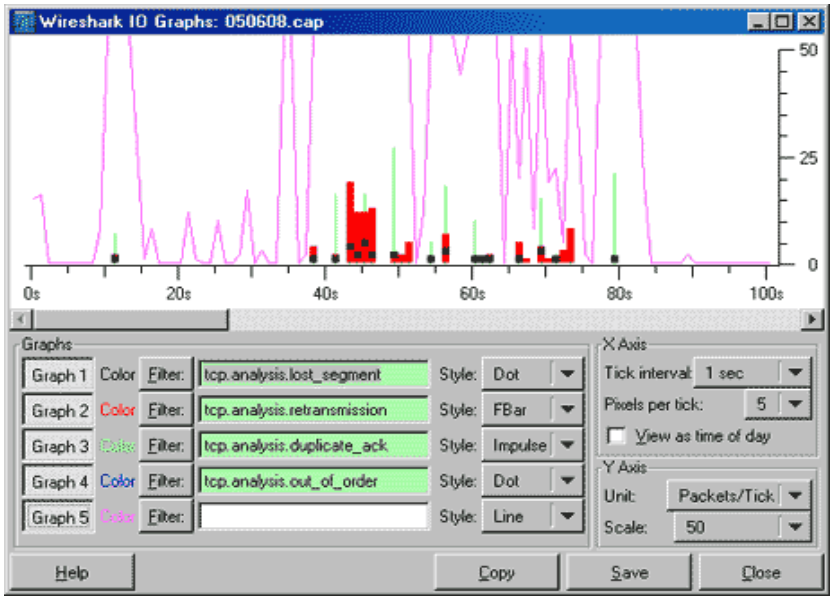


Figura 1: Captura de pantalla de Wireshark

Ettercap

És un sniffer per xarxes conmutades “Open Source”. Suporta direccions actives i passives de diferents protocols (inclús xifrats com SSH i HTTPS). Fa possible la injecció de dades a una connexió establerta. És capaç de filtrar mantenint la connexió gracies a què pot establir un Atac Man-in-the-middle(Spoofing) o enverinament de taules ARP (arp Spoofing). Aquest sniffer té com a llibreries base libpcap, libnet, libpthread i Zlib. També és capaç de detectar un sistema operatiu remot. Ettercap genera un fitxer log on guarda, en format binari, les captures realitzades.

Per a la majoria de les seves funcionalitats cal instal·lar el plugin necessari.

El aspectes més importants són:

- Injecció de dades a una connexió establerta emulant comandes o respostes mentre la connexió està activa.
- Compatibilitat HTTPS. És a dir pot interceptar connexions HTTP SSL, suposades segures, inclús si són a través d'un proxy.
- Compatibilitat SSH. És a dir pot interceptar usuaris i contrasenyes en connexions SSH.
- Intercepta tràfic remot a través d'un túnel GRE.

TCPDump

És un sniffer per xarxes TCP/IP. Monitoritza els paquets que entren i surten de la interfície de xarxa i els presenta en format llegible. Permet utilitzar filtres mitjançant un expressió de busca, de manera que tan sols mostra els paquets la capçalera dels quals coincideix amb ella. També pot ser utilitzada per a l'enginyeria inversa de protocols de xarxa. Per poder capturar tot el tràfic dirigit a la interfície cal col·locar-l'ho en mode promiscu. Pot capturar paquets en xarxes amb enrutaments no intel·ligents com podria ser un swicth o un router. En xarxes WI-FI pot capturar tots els paquets que arribin a l'antena i siguin reconeguts, de qualsevol xarxa disponible i fins i tot de varis canals simultanis, sempre que siguin adjacents i que la targeta no sigui molt selectiva en freqüència. Utilitza la llibreria pcap.

Win Sniffer

Com tots els sniffers serveix per administrar xarxes, però a diferència d'altres permet reconstruir el tràfic de la xarxa de manera simple i entenedora mentre altres sniffers tant sols mostren els paquets capturats. Pot capturar paquets en mode promiscu. A més a més pot descodificar FTP, POP3, HTTP, ICQ, SMTP, Telnet, IMAP i contrasenyes NNTP. Té un filtre per paquets molt intuïtiu. Té la possibilitat de ser

deixat dies sencers capturant tràfic ja que les captures les guarda en logs.

Darkstat

La diferència d'aquest sniffer en comparació amb d'altres és que un cop fetes les captures genera un informe en format HTML i permet realitzar estadístiques de direccions que es generen durant la comunicació entre hosts, del tràfic que es produeix i dels diferents port utilitzats per els diversos protocols.

Taffic-Vis

Aquest sniffer monitoritza tràfic TCP/IP i amb aquesta informació genera gràfics amb ASCII, HTML o Postscript. També permet analitzar tràfic entre hosts per determinar quins hosts s'estan comunicant i el volum de la comunicació. Utilitza com molts d'altres sniffers la llibreria libpcap.

Kismet

Aquest sniffer funciona amb qualsevol targeta càmbrica que suporti monotorització raw i pot capturar tràfic 802.11b, 802,11a i 802.11g. Es diferència d'altres sniffers inalàmbrics per el seu funcionament passiu. És a dir, que captura tràfic sense enviar cap paquet detectable, permeten detectar la presència de diferents punts d'accés i clients associats uns amb els altres.

Trobem tres parts diferenciades:

- Conté una sonda que pot ser utilitzada per recollir paquets.
- Un servidor que es pot utilitzar juntament amb la sonda o amb ell mateix extrapolant la informació i organitzant-la.
- El client el qual es comunica amb el servidor i mostra la informació que prèviament a recollit el servidor.

Hunt

És un altre exemple d'sniffer que funciona amb xarxes ethernet. Hunt examina la xarxa per certes signatures i diferents patrons en el tràfic els quals indiquen un esdeveniment concret o una condició. Quan reconeix una signatura hunt entra a la sessió i dóna accés a una connexió establerta la qual pot ésser utilitzada per explorar la xarxa.

Hunt té quatre trets distintius:

- Permet especificar quina és la connexió que vols capturar enlloc de haver de veure-ho tot en un log.
- Detecta un connexió ja establerta.
- Ofereix spoofing tools.
- Ofereix capturar sessions actives.

LinSniffer

És un sniffer molt simple l'objectiu principal del qual és capturar noms d'usuari i passwords. De fet la informació que mostra és perfecte per veure passwords i noms d'usuari però no està preparat per un anàlisi més detallat.

IPTraf

És un monitor de xarxa que genera gran varietat d'estadístiques com informació TCP, contador UDP, informació ICMP i OSPF, estat dels nodes, errors IP, etc.
També conté filtres per a diferents protocols per tal de veure tan sols la informació que ens interessa.

Un exemple de la interfície que té IPTraf es pot veure en la següent figura (Figura 2) :

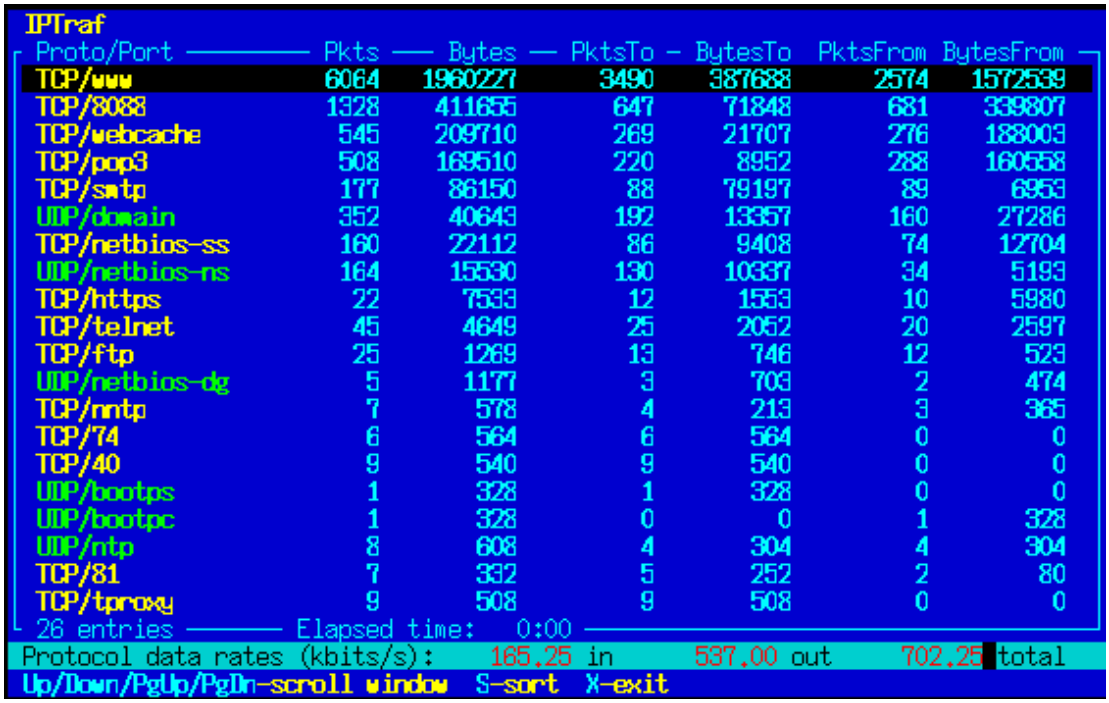


Figura 2: Captura de pantalla de IPTraf

Snort

És un sniffer/logger que serveix per detectar intrusions i atacs del tipus búfer overflows, CGI, SMB, escaneig de ports, etc. És capaç d'enviar alertes en temps real, enviant-les directament a l'arxiu de Unix syslog o fins i tot a un sistema Windows mitjançant SAMBA. Disposa d'una gran quantitat de filtres i patrons determinats. També disposa d'actualitzacions constants per fer front a atacs, escombrades o vulnerabilitats que es vagin detectant. Podem veure per consola els paquets que es van registrant i guardar-los en logs.

Altres sniffers:

Linux_sniffer: És semblant a LinSniffer. Dóna una sortida una mica més detallada què en el cas de Linsniffer i és molt fàcil d'utilitzar.

NFR: Network Flight Recorder és un sniffer comercial per detectar intrusions als sistemes.

QueSO: Serveix per a esbrinar quin Sistema operatiu corre en una màquina remota, analitzant les respostes TCP.

IPLog: És un logger de tràfic TCP/IP, UDP i ICMP el qual detecta escaneigs i possibles atacs per a sistemes Unix.

Cerberus Internet Scanner: CIS és un escanner el qual ajuda als administradors de xarxa de windows a detectar i corregir forats de seguretat. És capaç de generar documents en format HTML dels resultats de l'escaneig.

Retina: És un conegut escanejador de vulnerabilitats de sistema. Inclou la manera d'arreglar-los. Inclou moltes funcionalitats com pots ésser generació de reports o la detecció de
sistemes operatius remots.

Vetescan: És un escanejador de vulnerabilitats que té programes per comprovar i explorar errors de windows i Unix i corregir-los. Té la habilitat de poder saber quins serveis estan corrent a cada un dels hosts remots i generar gràfics de la xarxa.

Cheops: Programa que serveix per mapejar xarxes locals i remotes. Mostra el Sistema Operatiu de les màquines de la xarxa. Té la habilitat de poder saber quins serveis estan corrent a cada un dels hosts remots i generar gràfics de la xarxa.

Ngrep: Programa sensible a pcap el qual permet especificar expressions regulars esteses contra la carga de dades dels paquets. Reconeix TCP, UDP i ICMP per xarxes ethernet a través de PPP, SLIP i interfícies nul les.

Sam Sapade: És un programa online per investigar un direcció IP i trobar spammers.

Dsniff: És un sniffer per buscar paswords i més informació d'una xarxa incluint sofisticades tècniques per defensar la protecció dels switchers de la xarxa.

A la Figura 3 es pot veure una taula amb les principals característiques dels sniffers:

Figura 3: Taula resum dels diferents sniffers

Sniffers	LL	TXCT	PA	LLB	IDC	PE	FC	DA	DSO	IG	GC	GD	GG	MP
Wireshark	GPL	Ethernet	Més de 480	Libpcap	Si	Windows Mac Linux	Si	Si	No	Si	Si	Si	Si	P/NP
Ettercap	OS	LAN	Molts	Libpcap, libnet, libpthread , Zlib	No	Windows Mac Linux	Si	Si	Si	Si	Si	No	No	P/NP
TCPDump	OS	TCP/IP		libpcap	No	Linux Mac	Si	Si	No	No	No	No	No	P/NP
Win Sniffer	Shareware	TCP/IP	Ftp,pop3 http,icq smtp,telnet imap,nntp	winpcap	No	Windows	Si	No	No	Si	Si	No	No	P/NP
Darkstat	LGPL	TCP/IP	http	libpcap	No	Linux Mac	No	No	No	Si	Si	Si	Si	P/NP
Taffic-Vis	BSD	TCP/IP	___	libpcap	No	linux	No	No	No	Si	Si	Si	Si	P/NP
Kismet	GPL	LAN	802.11b 802,11a 802.11g	___	No	Linux Mac	No	Si	No	Si	Si	No	Si	__
Hunt	GPL	ethernet	Tcp udp arp, icmp	GlibC	No	Linux	Si	Si	No	Si	Si	No	No	__

Sniffers	LL	TXCT	PA	LLB	IDC	PE	FC	DA	DSO	IG	GC	GD	GG	MP
LinSniffer	GPL	Ethernet	__	___	No	Linux	No	Si	Si	No	No	No	No	__
Linux_sniffer	GPL	Ethernet	__	___	No	Linux	No	Si	Si	No	No	No	No	__
NFR	OPL	LAN	__	___	Si	Windows	Si	Si	No	Si	Si	Si	Si	P/NP
QueSO	GPL	TCP/IP	__	___	No	Linux	No	No	Si	No	No	No	No	__
IPLog	GPL	LAN	tcp/ip icmp udp	___	No	Linux	Si	Si	No	No	No	No	No	P/NP
IPTraf	GPL	LAN	Tcp icmp udp ospf	ncurses	No	Linux	Si	No	Si	Si	Si	Si	No	NP
Cerberus Internet Scanner	GPL	LAN	Tcp smtp ftp pop3	___	No	windows	No	Si	No	Si	Si	Si	No	NP
Retina	Shareware	LAN	___	___	No	Windows	No	Si	Si	Si	Si	Si	Si	__
Vetescan	GPL	LAN	___	___	No	Windows linux	No	Si	Si	No	No	No	No	__
Cheops	GPL	LAN	___	___	No	Windows linux	No	Si	Si	Si	Si	Si	Si	__
Ngrep	BSD	Ethernet	Ipv4,icmp tcp,udp	libpcap	No	Linux Mac,windows	Si	No	No	Si	Si	Si	No	__

Sniffers	LL	TXCT	PA	LLB	IDC	PE	FC	DA	DSO	IG	GC	GD	GG	MP
Sam Sapade	Free	Ethernet	__	___	No	All	No	No	No	No	No	No	No	__
DSniff	Freeware	LAN	__	___	No	windows	Si	Si	No	No	Si	Si	No	__
Snort	GPL	LAN	__	___	No	Linux	Si	Si	No	N	Si	Si	No	__

Llicencia: LL
Tipu de xarxes de les que pot capturar tràfic: TXCT
Protocols que admet: PA
Llibreria base: LLB
Injecció dades a la connexió: IDC
Plataforma d'execució: PE
Filtrar captures: FC
Detecció atacs: DA
Detecció sistema operatiu: DSO
Interfície gràfica: IG
Guardar captures: GC
Generació de documents: GD
Generació de gràfics: GG
Mode Promiscu/No promiscu: MP

2.5 Tecnologies de distribució

Abans de començar a explicar les diferents tecnologies de distribució farem un breu comentari sobre què significa Sistemes de distribució.

2.5.1 Introducció

Un sistema distribuït es defineix com: una col·lecció de computadores separades físicament i connectades entre elles per mitjà d'una xarxa de comunicacions distribuïda. Cada màquina té els seus propis components de hardware i software i l'usuari no percep quan accedeix a recursos remots, a recursos locals o a un grup de computadors els quals utilitzen un software per aconseguir un objectiu comú.

L'avantatge dels sistemes distribuïts és que quan un component del sistema es descompon un altre component pot realitzar la seva funció. Això s'anomena “Tolerància a Fallades”.

El tamany del sistema pot ser molt variat, des de desenes de hosts(xarxa d'àrea local) i centenars de hosts(xarxa d'àrea metropolitana), fins a milions de hosts (Internet).

Algunes de les característiques més importants a les quals vol arribar un sistema distribuït són:

- Per als usuaris és similar al Sistema Centralitzat.(Un computador).
- S'executa a múltiples computadors.
- Té varies còpies de mateix Sistema Operatiu o de diferents Sistemes Operatius que donen els mateixos serveis.
- Té transparència. (La utilització de múltiples processadors i l'accés remot és invisible)

Un invocació es composa en general dels següents passos:

- Encapsulat dels paràmetres (Utilitzant la funcionalitat de serialització)
- Invocació del mètode (Sempre ho realitza el client). L'invocador, per tant, es queda normalment esperant una resposta.
- Quant el servidor acaba la execució serialitza el valor de retorn (si existeix) i l'envia al client.
- El client continua la seva execució com si la invocació hagués estat local.

2.5.2 Història

La base de les aplicacions distribuïdes són els ordinadors i concretament l'aparició d'internet. L'origen el podem trobar a la tesi doctoral de Leonard Kleinrock, en la qual es van establir els principis de les xarxes de paquets i que va servir per al desenvolupament de l'ARPNET. Tot i així va ser Bob kahn qui va tenir el paper més important en el desenvolupament d'ARPNET, al voltant de 1969.

De totes maneres no va ser fins a l'aparició de TCP/IP, dissenyat per Bob Kahn i Viton Cerf, i amb ell la interconnexió de xarxes el 1974 que les xarxes distribuïdes van agafar un caire més important el concepte de xarxes distribuïdes.

El 1983 es defineix el protocol TCP/IP com a protocol oficial i sorgeixen les primeres xarxes aplicacions distribuïdes.

El model dominant a les xarxes de comunicacions era, i és, client-servidor, especialment útil per a la comunicació d'ordinadors entorn el qual es construeix una entrada i una sortida. El 1985 va sortir un article de Birrell i Nelson en el qual es deia que es deixava a un programa cridar procediments localitzats en altres ordinadors. D'aquesta manera el pogramador es preocupa tan sols de la E/S. Aquest mètode es coneix com (RPC).

Les tecnologies actuals es basen en el middleware. Aquest el podríem definir com una capa software entre els processos d'usuari i els del sistema operatiu. Per tant per al programador queda amagada la complexitat del sistema operatiu. Dins aquesta tecnologia trobem middleware orientat a objectes la qual a desenvolupat Sun Microsistems i la middleware orientat a components desenvolupada per CORBA:OMG.

2.5.3 RPC

RPC prové de l'anglès “Remote Procedure Call”. És un protocol que permet a un programa executar codi a una màquina remota sense preocupar-se per la comunicació entre aquestes. RPC va ser un gran abans ja que utilitzant sockets el programador havia d'estar pendent de les comunicacions. Les RPC són molt utilitzades en entorns client-servidor. Essent el client qui inicia una petició perquè s'executi cert codi en el servidor.

Existeixen diferents tipus de RPCs, molts d'ells estandarditzats com pot ser l'RPC de Sun, ONC RPC, l'RPC de OSF anomenat DCE/RPC i el model d'Objectes de Components de Microsoft DCOM. Tot i això cap d'aquests models és compatible entre si. La majoria d'ells utilitzen un llenguatge de descripció d'interfície IDL que

defineix els mètodes exportats per el servidor.

Avui en dia és molt utilitzat l'XML com a llenguatge per definir IDLs i HTTP com a protocol de xarxa.

RPC de Sun

Un de RPC podria ser el RPC de Sun. Aquest proporciona un llenguatge de definició d'interfície anomenat XDR(eXternal Data Representation) i un compilador d'interfícies anomenat rpcgen. A partir de la interfície definida amb XDR i del compilador rpcgen s'obtenen la majoria dels components d'una RPC. És a dir:

- L'stub del client.
- El programa principal del servidor.
- Els fitxers de definició o de capçaleres de l'stub del client i les rutines de servei del servidor.
- Les rutines de serialització de dades.

A la Figura 4 es pot veure un esquema del funcionament del RPC.

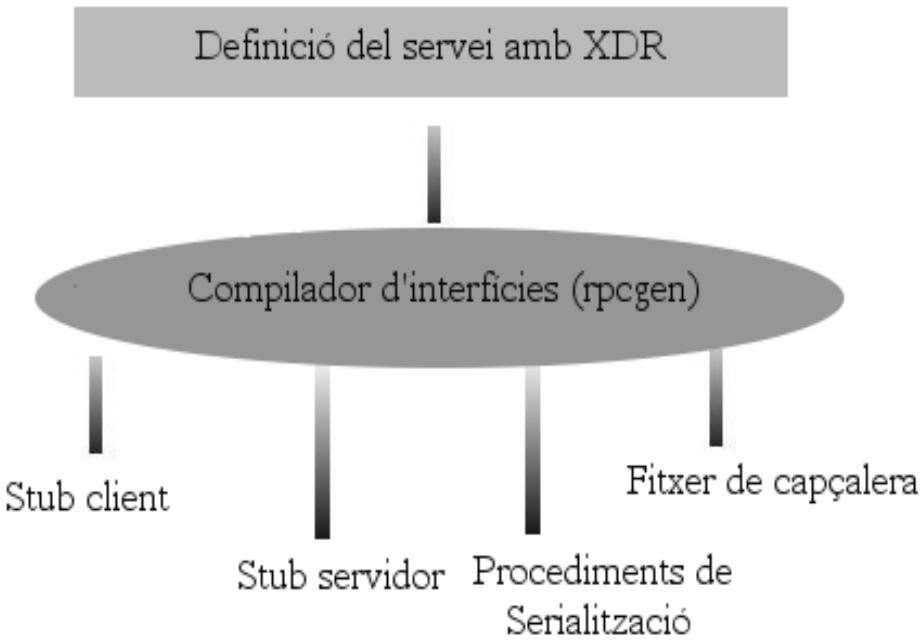


Figura 4: Parts d'un RPC

El llenguatge XDR es pot utilitzar no solament per especificar les operacions que ofereix el servidor, sinó també el seu identificador o número de servei i la seva versió.

Aquest llenguatge ofereix una notació semblant a C. Aquest RPC pot passar qualsevol estructura de dades com arguments o resultats i disposa d'unes rutines estàndard de serialització de dades.

El RPC de Sun no ofereix un servei de binding a nivell de xarxa, sinó local. És a dir que a cada màquina hi ha un binder. Degut a això aquest RPC no ofereix transparència total, de tal manera que l'usuari ha de realitzar petites modificacions a les crides del seu programa.

2.5.4 RMI

RMI significa Java Remote Method Invocation i és un mecanisme que ofereix Java per invocar un mètode de forma remota. És un simple mecanisme per a la comunicació de servidors dins aplicacions distribuïdes basades exclusivament amb Java. Per a la comunicació entre altres tecnologies podem utilitzar CORBA ò SOAP.

RMI es caracteritza per la seva facilitat d'utilització. El gran avantatge és que permet referència a objectes (no permès per SOAP), recol·lecció de brossa distribuïda (Garbage Collector distribuït) i pas de tipus arbitraris (funcionalitat que no ofereix CORBA).

Mitjançant RMI, un programa Java pot exportar un objecte. Això vol dir que aquest queda accessible a través de la xarxa, mentre que el servidor queda a la espera de peticions a un port TCP. Un client, per tant, pot agafar l'objecte de la xarxa i invocar-ne els seus mètodes. Un esquema de les parts d'un sistema RMI el podem veure a la Figura 5.

Des de l'exterior, l'estat d'un objecte només es pot canviar a través de la crida de certs mètodes de l'objecte, definits com a públics, per aquest motiu es diu que un objecte és una entitat encapsulada.

Java té dos tipus d'objecte: els locals i els remots.

Un objecte és local si els seus mètodes es invocar dins de la seva màquina virtual. És a dir, per el procés que va crear l'objecte o per threads d'aquest.

Un objecte és remot si els seus mètodes es poden invocar per processos que es troben a altres màquines virtuals.

A Java també trobem dos tipus de interfícies. La local, accessible per processos dins de la mateixa màquina virtual, i la remota la qual recull la declaració de les operacions que formen el servei. A diferència de CORBA els objectes que es passen

només són d'entrada.

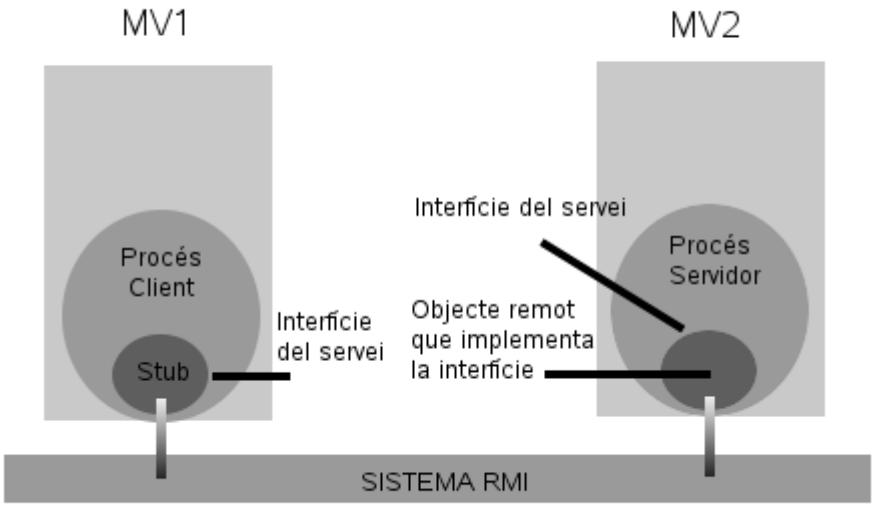


Figura 5: Parts d'un sistema RMI

2.5.5 CORBA

CORBA significa Common Object Request Broker Arquitecture i és un model de suport de programació distribuïda. Corba és simplement un model que facilita el desenvolupament i la interoperabilitat de sistemes distribuïts orientats a objectes.

El model de gestió d'objectes distribuïts de Corba es composa de: (en trobem un esquema a la Figura 6).

- El model d'objectes a on es descriu què és un objecte CORBA.
- El model de referència a on s'explica quina arquitectura es proposa per permetre que els objectes Corba es relacionin.

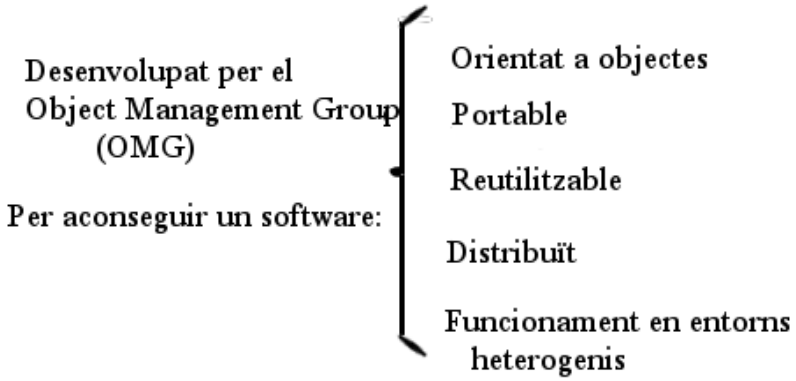


Figura 6: Principals característiques de CORBA

Corba és un model client/servidor orientat a objectes. Un servidor Corba ofereix serveis a través d'un objecte CORBA.

Corba utilitza un llenguatge de definició d'interfícies anomenat IDL (Interface Definition Language) i defineix la interfície que ofereix l'objecte. Aquesta interfície conté el nom del servei i les firmes de cada procediment indicant el nom de la operació, el conjunt de paràmetres d'entrada (in), de sortida (out) o d'entrada/sortida, amb els seus respectius tipus i el tipus del resultat de la operació. També es poden associar excepcions d'usuari o del sistema a les operacions.

Un objecte Corba s'identifica mitjançant la seva referència a objecte. Aquest identificar serà utilitzat pel client per comunicar-se amb el servidor.

La comunicació entre client/servidor és síncrona o bloquejant. Tot i això també es permet comunicació no bloquejant de dos tipus: asíncrona i en un sol sentit.

Per a la transmissió tan de la petició com de la resposta intervé la interfície de l'ORB. Aquest té dos tipus d'interfície: la pública que figura dins el model de l'OMG i la privada que és la que generant els diferents fabricants per permetre la connexió entre els stubs i l'ORB.

A la Figura 7 podem veure l'estructura client/servidor:

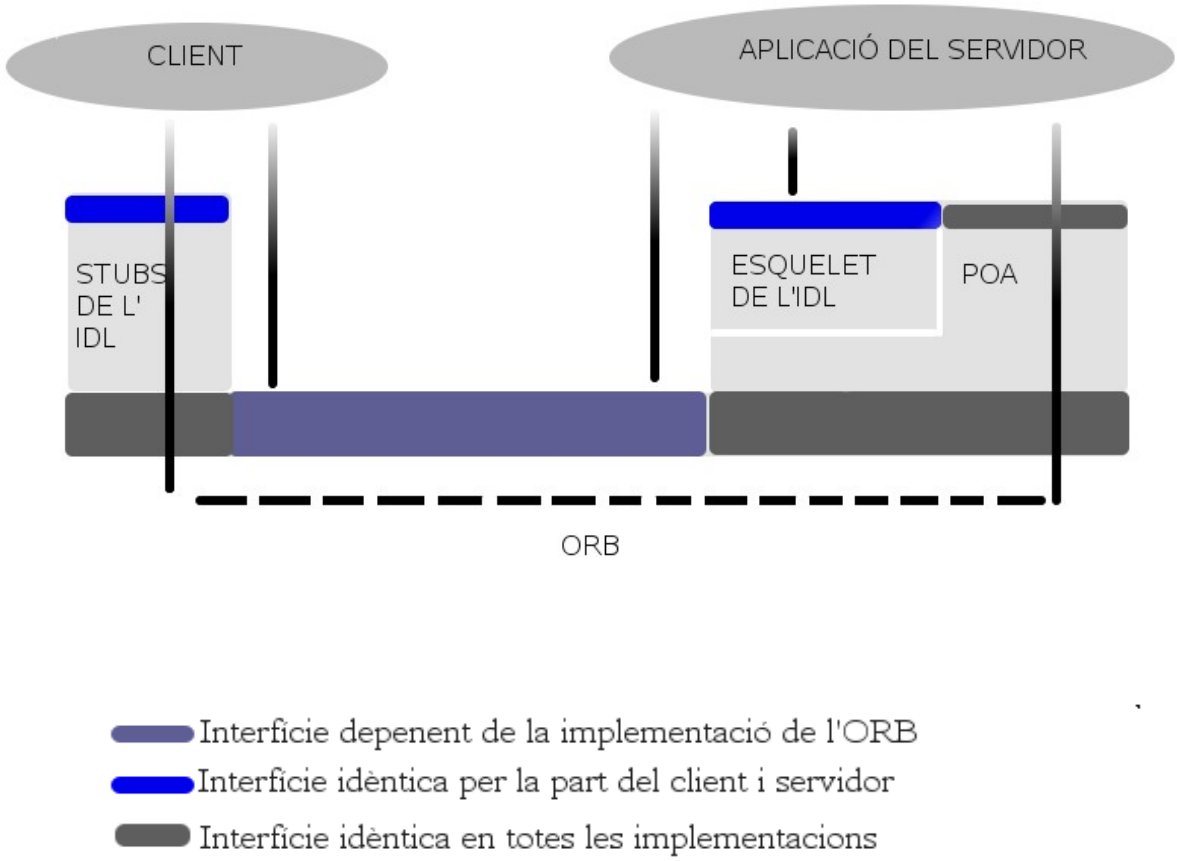


Figura 7: Estructura client/servidor CORBA

Corba ofereix un conjunt d'objectes anomenats facilitats comunes. Aquests són serveis útils per qualsevol entorn.

Finalment, de l'arquitectura CORBA podem treure les següents conclusions:

- Permet als programadors no construir certes eines necessàries.
- Si el conjunt de serveis i facilitats estan disponibles en més entorns, les aplicacions es poden moure d'un entorn cap a un altre.
- El client coneix com cridar operacions d'interfícies estàndards sobre qualsevol plataforma. Això vol dir que es poden crear sistemes oberts.

2.5.6 J2EE

J2EE és una plataforma de programació per desenvolupar i executar software d'aplicacions amb llenguatge de programació JAVA amb arquitectura de N nivells distribuïda, basat amb components de software modulars executant-se en un servidor d'aplicacions. Aquesta plataforma està definida per una especificació, la qual és considerada informalment com un estàndard ja que els subministradors han de complir certs requisits de conformitat.

J2EE inclou diverses especificacions d'API com podrien ser JDBC, RMI, e-mail, JMS, Serveis web o XML. J2EE també configura algunes especificacions úniques. Alguns exemples podrien ser Enterprise JavaBeans, servlets, portlets, JSP i diverses tecnologies de serveis web.

Altres avantatges d'utilitzar J2EE són, per exemple, que el servidor d'aplicacions pot manejar transaccions, la seguretat, la escalabilitat, concurrència i gestió dels components desplegats, cosa que vol dir que els programadors es poden concentrar més amb la lògica de negoci enlloc de preocupar-se per temes de baix nivell.

Cal comentar també que un dels grans avantatges de J2EE és que es pot començar a desenvolupar una aplicació sense cap cost.

2.5.7 Serveis Web

Un servei web és un conjunt de protocols i estàndards que serveixen per intercanviar dades entre diferents aplicacions. Aquestes aplicacions poden estar desenvolupades amb llenguatges de programació diferents, i executades sobre plataformes també diferents. Un exemple podria ser intercanviar dades a xarxes d'ordinadors com internet. La seva interoperabilitat es deu a la utilització d' estàndards oberts.

OASIS i W3C s'encarreguen de l'arquitectura i la reglamentació dels serveis web. Per altre banda trobem l'organisme WS-I el qual s'encarrega de definir de manera més exhaustiva aquests estàndards.

Els estàndards utilitzats són: Web Services Stack, XML, SOAP o XML-RPC, WSDL, UDDI, WS-Security i OASIS.

Pel què fa a als avantatges d'utilitzar Serveis Web, les més importants són:

- Aporten interoperabilitat entre les aplicacions de software independentment de les seves propietats i de les plataformes s'obre les quals es troben instal·lats.

- Els serveis web fomenten els estàndards i protocols basats amb text i d'aquesta manera és més fàcil accedir al seu contingut i entendre el seu funcionament.
- Es poden aprofitar dels sistemes de seguretat firewall sense necessitat de canviar les regles de filtrat ja que es recolzen s'obre HTTP.
- Permeten combinar fàcilment serveis i software de diferents companyies per donar serveis integrats.
- Permeten interoperabilitat entre plataformes de diferents fabricants.

Els serveis web, però, també tenen algun inconvenient:

- Per realitzar realitzar transaccions no es poden comparar pel què fa al seu grau de desenvolupament amb els estàndards oberts de computació distribuïda com CORBA.
- El seu rendiment és baix si es compara amb RMI o CORBA.
- Com que es recolzen amb HTTP, poden esquivar mesures de seguretat basades amb firewall.

Tot i això la principal raó per utilitzar Serveis web és que es basen amb HTTP sobre TCP al port 80. Com que les organitzacions protegeixen les seves xarxes per mitjà de firewalls, tanquen quasi tots els ports menys el port 80,que és, precisament, el que utilitzen els navegadors.

Una altre raó per la qual s'utilitzen els Serveis web és perquè abans que existís SOAP, no hi havia bones interfícies per accedir a funcionalitats d'altres ordinadors de la xarxa.

Una altre bona raó és la seva flexibilitat. Aporten independència entre l'aplicació que utilitza el servei web i el propi servei. Així els canvis en un no han d'afectar a l'altre. Això és bo ja que cada cop més es tendeix a construir grans aplicacions a partir de components més petits.

2.6 Tecnologies web

2.6.1 Struts

Struts és una eina de suport per a desenvolupar aplicacions web sota el patró MVC (Model Vista Controlador) i sota la plataforma J2EE. Struts permet reduir el temps de desenvolupament d'una aplicació.

Una aplicació web no són només pagines html. Sinó que pot produir respostes dinàmiques. Una aplicació web pot tenir interacció amb bases de dades i lògica de

negoci. Per aquest motiu es va crear el Model Vista Controlador. Així doncs es complica forca la feina del programador i per aquest motiu apareix struts.

Struts conté en un simple fitxer “struts-config.xml” l'enrutament i el mapping de totes les accions de l'aplicació.

Struts dóna al programador tres components clau:

- Una “request handler” donada per l'acció mapejada a una URI.
- Una “response handler” la qual traspassa el control a un altre recurs que completa la resposta.
- Una “tag” llibreria que ajuda als programadors a crear de manera interactiva formularis a JSP.

2.6.2 JSP

JSP és una tecnologia que proporciona una manera ràpida de crear pàgines web amb contingut dinàmic. En el fons una JSP és un fitxer de text que té contingut estàtic expressat amb format HTML, SVG, XML o WML i elements JSP amb contingut dinàmic.

Una JSP és traduïda a un servlet abans de ser executada, processa una request HTTP i genera una response igual que qualsevol servlet.

Les JSP contenent XML tags, molt útils per estalviar temps en la programació, i codi javascript per fer més fàcil certes parts del codi. Tot i això el javascript canvia segons el navegador i esdevé una potencial font d'error en aplicacions web.

Una JSP agafa els tags XML i els transforma en codi estàtic HTML i després és compilada per transformar-la en un servlet, per tant, és lògic pensar que necessita la Màquina Virtual de Java per funcionar. També cal dir que la JSP compilada es guarda a la memòria del servidor i quan torna a ser cridada el temps d'execució millora notablement.

2.6.3 AJAX

AJAX acrònim de Asynchronous Javascript And XML, és un tècnica de desenvolupament web per crear aplicacions interactives o RIA (Rich Internet Applications). AJAX s'executa en el client, és a dir, en el navegador de l'usuari mentre es manté una comunicació asíncron amb el servidor a segon pla. El gran avantatge d'AJAX és que permet realitzar canvis a les jsp sense necessitat de recarregar-la, el que significa augmentar la interactivitat, la velocitat i la utilitat de

les aplicacions.

Ajax és una tecnologia asíncrona, en el sentit que les dades addicionals es demanen al servidor i es carreguen en segon pla sense interferir amb la visualització ni el comportament de la JSP. JavaScript és el llenguatge que normalment s'utilitza per realitzar les funcions que realitzen les crides d'AJAX, mentre que l'accés a les dades es realitza mitjançant XMLHttpRequest. De tota manera, no és necessari que el contingut estigui en format XML.

Ajax és una tècnica vàlida per moltes plataformes i utilitzable per molts sistemes operatius i navegadors ja que està basat amb estàndards oberts com Javascript i Document Object Model.

Ajax és una combinació de quatre tecnologies ja existents:

- XHTML (o HTML) i fulles d'estils CSS per el disseny que acompanya l' informació.
- Dovument Object Model (DOM) accedit amb un llengutatge d'scripting per part de l'usuari, especialment implementacions ECMAScript com Javascript y Jscript, per mostra i interactuar dinàmicament amb la informació presentada.
- L'objecte XMLHttpRequest per intercanviar dades de forma asíncrona amb el servidor web.
- XML és el format utilitzat normalment per la transferència de les dades sol·licitades al servidor. Tot i que qualsevol tipus de format pot funcionar, incluint HTML preformatejat, text pla, JSON i fins i tot EBML.

3 Anàlisi dels requisits

3.1 Requisits funcionals

Abans de començar a nombrar requisits funcionals faré una breu definició. Un requisit funcional defineix el comportament intern del software: càlculs, detalls tècnics, manipulació de dades i altres funcionalitats específiques que mostren com els casos d'ús seran portats a la pràctica.

El primer requisit que m'agradaria anomenar és **realitzar captures.** És el requisit més important i per a complir-l'ho caldrà connectar les tres parts bàsiques de l'aplicació: l'aplicació en si mateixa, la qual podríem anomenar capa de presentació, amb l'ejb, el qual podria ser la capa de negoci de l'aplicació, i aquest amb la capa més interna que correspon al servidor de captures.

Hi ha 4 requisits els quals són: guardar captures, configurar noves captures, obrir captures i borrar captures. Aquests tindrien tots el mateix grau d'importància.

Guardar Captures: Haurà d'existir un botó a la barra del menú superior per realitzar aquesta acció. Per complir aquest requisit caldrà fer que l'aplicació guardi les captures que es troben amb format XML. Caldrà utilitzar la llibreria JDOM per tractar les captures i transformar-les a un arxiu XML, jeràrquic.

Configurar noves captures: Haurà d'existir un botó a la barra del menú superior per realitzar aquesta acció. És important poder tornar configurar una captura en qualsevol moment i no només quan entrem a l'aplicació. Caldrà modificar els paràmetres de la request per tal que el servidor utilitzi els nous valors.

Obrir captures: Haurà d'existir un botó a la barra del menú superior per realitzar aquesta acció. Aquest requisit és necessari per complementar el requisit de guardar les captures. També caldrà utilitzar JDOM per tractar les captures i posar-les a la request per tal què la JSP la pugui entendre.

Borrar captures: Haurà d'existir un botó a la barra del menú superior per realitzar aquesta acció. Aquest requisit és indispensable ja que és necessari netejar la taula de captures després de varies captures. Tan sols cal netejar la request de captures i així la llibreria displaytag no mostrarà captures.

Requisits Funcionals (Figura 8):

Nom	Requisit Usuari	Conseqüència tècnica
RC	Realitzar Captures	Utilització d'un ejb3 el qual s'encarrega de demanar i processar les captures al servidor C++.
GC	Guardar captures	Aplicació de Jdom per creació de fitxers xml. Utilització de formularis HTML per passar la informació a l'action a través de la “request”.
CNC	Configurar noves captures	Aplicació de divs ocults. El qual es torna visible quan premem l'acció concreta al menú. Utilització formularis HTML per passar la informació a l'action a través de la request
OC	Obrir captures	Aplicació de Jdom per a la lectura de fitxers xml. Utilització de formularis HTML per passar la informació a l'action a través de la “request”. Utilització de DisplayTag per mostrar les captures.
BC	Borrar captures	Crida a l'action, sense formulari. Passant paràmetre per el GET per indicar que ha de borrar captures de la sessió.
ASX	Accés a altres sniffers de la xarxa	Crida al servidor de captures. Enllaç linkable a la url en qüestió.

Figura 8: Taula de requisits funcionals

3.2 Requisits no funcionals

Abans de començar a nombrar **requisits no funcionals** en donaré una breu definició. Un requisit no funcional és un requisit que especifica criteris que es poden utilitzar per jutjar l'operació d'un sistema enlloc del seu comportament específic.

L'aplicació ha de ser **ràpida** per tant s'haurà d'utilitzar tecnologies que permetin aquest comportament com poden ésser ajax per tal de no recarregar la pàgina cada cop que s'accedeixi al servidor. La utilització d'un ejb que corri en una màquina virtual diferent de la de l'aplicació en si mateixa per tal de repartir càrrega de treball. L'aplicació haurà d'estar corrent en un ordinador Linux i amb la llibreria libpcap. La llibreria accedeix al kernel de la màquina i és aquest qui realment realitza les captures. Això fa que l'aplicació sigui més ràpida i eficient que no pas en sistemes windows. En aquest la captura i el filtratge es realitza des de la llibreria.

L'acceptació també ha de tenir una **Utilització Intuïtiva** per tant serà imprescindible la utilització d'uns bons css i la utilització de “divs” ocults. Per tal de què els css tinguin la seva funció alhora de complir aquest requisit també serà imprescindible un establiment de funcions ben definit.

Lògicament, l'aplicació ha de quedar **ben presentada**. Per a complir aquests requisit caldrà definir correctament l'estructura de la pàgina JSP i igual què en el requisit anterior utilitzar uns bons css.

L'aplicació també haurà de ser **efectiva.** Per a complir aquest requisit caldrà realitzar un bon control d'errors i preveure possibles errades que es puguin donar en temps d'execució. També caldrà desenvolupar un codi net, sense cicles i sense redundàncies.

L'aplicació haurà d'ésser en un entorn web per tal de complir el requisit de **no presència física**. L'usuari no haurà d'estar davant de la màquina per realitzar captures. Simplement cal tenir internet per connectar-se a l'aplicació.

Requisits no Funcionals(Figura 9):

Nom	Requisit Usuari	Conseqüència tècnica
R	Rapidesa	Utilització ajax, ejbs, dues JVM
UI	Utilització intuïtiva	Utilització bons css i utilització de divs ocults. Establiment de funcions ben definides.
NPF	No presència física	Desenvolupament de l'aplicació en un entron web.
ABP	Aplicació ben presentada	Utilització d'una estructura de la pàgina JSP ben definida. Utilització d'uns bons css.
E	Eficiència	Realitzar un bon control d'errors, un codi net, sense cicles i sense redundàncies.

Figura 9: Taula de requisits no funcionals

4 Metodologia i planificació

4.1 Introducció (Metodologies d'enginyeria del software)

Si parlem de diferents metodologies de Software Enginiering trobem 4 fonamentals fases a quasi tots aquests mètodes. Aquestes són anàlisi, disseny, implementació i test. L' objectiu d'aquestes metodologies és adreçar sobre el què s'ha de fer i de quina manera s'ha de fer.

La fase d'anàlisi defineix els requeriments del sistema. Sense tenir en compte de quina manera seran aquests requeriments portats a terme. Bàsicament, en aquesta fase, es defineix el problema el qual s'està intentant resoldre. El resultat final d'aquesta etapa és un document de requeriments.

La fase de disseny s'estableix l'arquitectura de l'aplicació. Aquesta fase comença amb el document de requeriments fet a la fase anterior i mapeja aquests requeriments a arquitectura i tecnologies a utilitzar. Per tant, en aquesta fase es realitza un document arquitectònic, es desenvolupa un pla d'implementació, un document de prioritats critiques(les quals són essencials per a l'èxit del projecte), un document que parli de com ha de ser l'entorn i com s'ha de mostrar al client. Finalment, també ha de tenir pla de test en el qual es defineixen els tests que s'hauran de realitzar per tal de garantir la qualitat del producte.

La fase d'implementació és el moment de començar a desenvolupar codi i construir els diferents elements de l'aplicació seguint els anàlisis anteriors. En aquesta fase cal fer un llistat d'errors. Errors crítics per a l'aplicació, errors no crítics i errors desconeguts.

La última fase, no per això menys important, és la fase de Test. En aquesta fase confirmem la qualitat del producte entregat. Cal tenir en compte que és millor un producte amb poques funcionalitats però ben provat que no pas un amb moltes funcionalitats però errades que portin al no compliment dels requisits establerts a la primera fase.

Trobem diferents metodologies alhora de realitzar un projecte algunes d'aquestes les podem veure en el següent apartat.

4.2 Tipus de metologies

4.2.1 Metodologia seqüencial

En aquest tipus de metodologia, també anomenada com metodologia en cascada, les diferents fases del projecte van una seguida de l'altre. Així doncs, primer trobem la fase d'anàlisi, després trobem la fase de disseny, seguida de la implementació i per últim la fase de test.

Una metodologia seqüencial es sol utilitzar quan la complexitat del projecte és baixa i els requeriments són estàtics. Molts cop es comença a picar codi sense un bon analisis del projecte i després és necessari reescriure'l. Aquest contratemps el podem evitar utilitzant una metodologia seqüencial.

Tot i això hi ha algun desavantatge quan utilitzem una metodologia seqüencial. Cal conèixer perfectament les tecnologies aplicables a l'aplicació ja un cop realitzat l'anàlisi ja no hi ha marxa endarrera. Per aquest motiu un possible error a l'etapa d'implementació no deixaria refer l'anàlisi. Un esquema d'aquesta metodologia el podem veure a la Figura 10.

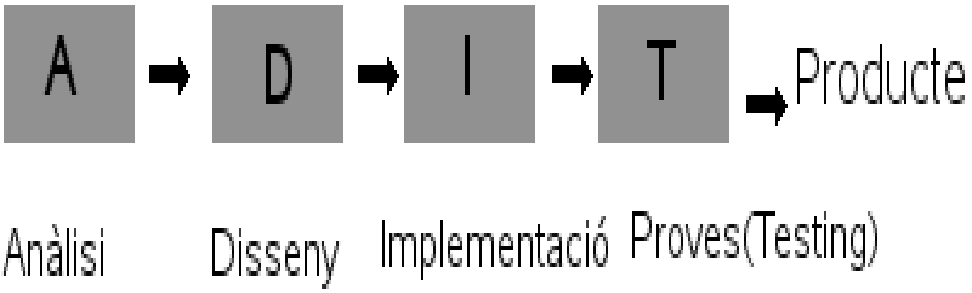


Figura 10: Esquema de la Metodologia seqüencial

4.2.2 Metodologia en cercle

Aquesta metodologia també es coneguda com metodologia amb espiral. Bàsicament, aquesta metodologia, fixa problemes provinents de l'aplicació de la metodologia seqüencial. Aquesta metodologia conté les 4 fases: anàlisi, disseny, implementació i

test.

A cada fase es dedica un petita porció de temps i es va iterant per cada una d'aquestes. Així, totes les parts van canviant a mesura que el projecte va avançant. Això permet que errades en els requeriments o errades enfront les tecnologies a utilitzar es puguin corregir. Cada una una de les fases s'intercomunica i interactua amb les altres per tal de millorar tots els aspectes possibles de l'aplicació.

Per altre banda, però, no hi ha un límit d'iteracions a realitzar sobre les 4 fases i és possible que cada cop un fase comporti més feina a una altre. Possiblement no s'arriba mai a una estructura vàlida del projecte.

En aquesta metodologia és molt important l'aprenentatge durant el desenvolupament, ja que així es pot aplicar a la següent iteració.

Un esquema d'aquesta metodologia el trobem representat a la Figura 11.

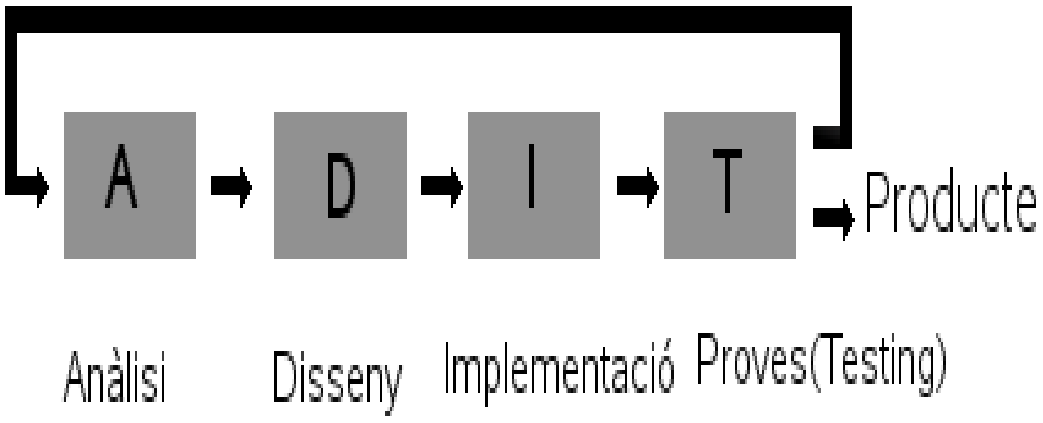


Figura 11: Esquema metodologia cercle

4.2.3 Metodologia incremental

Aquesta metodologia és semblant a la metodologia en cercle. Es va iterant sobre les 4 fases ja comentades anteriorment amb la diferència que a cada iteració es fa un petit increment en els requisits que hauria de complir el projecte.

4.2.4 Metodologia àgil

Aquesta metodologia és iterativa. No intenta minimitzar els canvis, sinó que està

preparada per acceptar-los. És una metodologia, per tant, adaptativa. Incorpora el feedback sobre el procés i prioritza la col·laboració amb el client. Prefereixen codi executable abans que una documentació exhaustiva. També es caracteritza per minimitzar l' overhead que es produeix en les iteracions.

4.2.5 Metodologia basada en prototips

És un model de desenvolupament que posa èmfasi a la etapa de Especificació de requeriments a través de la construcció de prototips que s'apropen a l'usuari a la idea final del sistema, amb l'objectiu de poder clarificar els requeriments.

La metodologia basada amb prototips consta diferents fases:

- Investigació preliminar: En aquesta fase es determina el problema i el seu entorn, la importància del problema i els possibles afectes sobre el projecte. En aquesta fase però també es busca una idea general de la solució.

- Definició dels requeriments del sistema: L'objectiu d'aquesta fase es registrar tots els requeriments de l'usuari. Aquesta fase és la més important ja que es determinen requisits mitjançant la construcció, demostració i retroalimentació del prototip.

- Disseny tècnic: Aquesta etapa conté dues parts molt importants, per un banda cal fer la documentació del disseny que explica com serà l'estructura del software. La segona part és la producció del tots els requeriments i assegurar un fàcil manteniment del software en el futur.

- Programació i prova: és a on es realitza la implementació seguint el disseny tècnic i provada per assegurar el bon funcionament.

- Operació i manteniment: En aquesta fase es tracta de portar els diferents prototips en un àmbit d'explotació.

- Anàlisi gros i especificacions: L'objectiu d'aquesta fase és desenvolupar un disseny bàsic per el prototip inicial.

- Disseny i construcció: L'objectiu d'aquesta fase és desenvolupar el prototip inicial.

- Evaluació: Aquesta fase conté dues parts molt importants. Per una banda extreure de l'usuari els requeriments addicionals del sistema i verificar que el prototip sigui correcte amb els requeriments. Si es troben errades, el prototip es

corregeix i reavaluat.

- Modificació:Aquesta fase es porta a terme quan els requeriments del sistema són alterats a la subfase l'avaluació.

- Termino: L'objectiu és establir aspectes de qualitat i de representació del sistema.

Un esquema d'aquesta metodologia el trobem a la Figura 12.

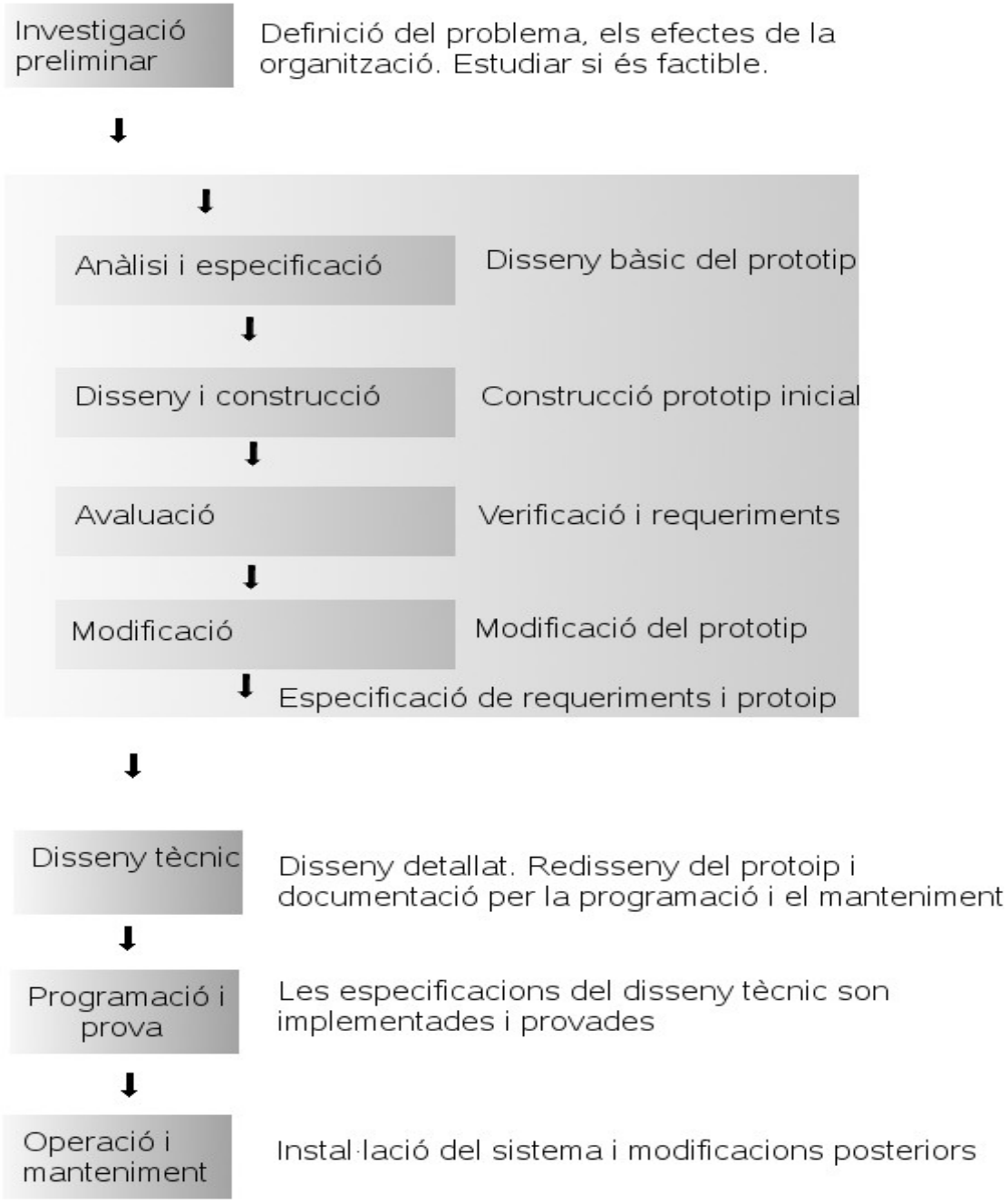


Figura 12: Esquema metodologia per prototips.

Els avantatges d'aquest sistema són la reducció del risc del projecte, reducció de temps i costs, increments en l'acceptació del nou projecte, millores pel què fa a l'administració, etc.

Algunes de les desavantatges són la dependència de les eines de software per a l'èxit, ja que la disminució de la incertesa depen de les iteracions realitzades i sovint es van utilitzant tecnologies més sofisticades el què fa a aquest procés dependent de les tecnologies utilitzades. A més l'usuari podria confondre el prototip amb el producte final i malinterpretar-l'ho.

4.3 Descripció metodologia escollida

La metodologia que vaig escollir alhora de començar el projecte va ser la metodologia incremental.

Tenia clar quin era l'objectiu final a assolir, però sovint quan pensava, abans de començar el projecte, com i de quina manera el faria no ho tenia molt clar. Doncs, hi havia moltes coses i molts paràmetres a tenir en compte. Hi havia temes que no s'havia com abarcar. Per aquesta motiu vaig pensar en anar assolint metes, objectius, més petits i més moderats que no semblessin inabastables, i, d'aquesta manera anar formant un fonament sobre del qual s'aniria formant l'aplicació. Les fites, objectius, que volia assolir havien de ser prèviament meditats, ja que una errada en aquesta increment dels requeriments podria portar-me al fracàs. Podia posar-me un fita, increment, massa alta i no poder-la assolir o per altre banda fites masses petites. Tant per un motiu com per l'altre, això significa, sens dubte, un retard en el projecte. Per aquest motiu, aquest, va ser un tema en el qual vaig rebre una valiosa ajuda per part de l'Isaac Gelado.

4.4 Planificació temporal (Diagrama de Gonn)

La planificació la vaig dividir en les següents fases:

T1. Baixar sniffer i provar-l'ho

Aquesta és la primera feina que s'ha de fer. Consisteix en baixar-me l'sniffer, aplicació d'escriptori, de “www.sourceforge.net”. Seguir les instruccions d'installació, les quals es troben a la carpeta doc/troff/install.t i finalment provar-l'ho i veure possibles problemes alhora d'executar-se.

T2.Fer funcionar l'sniffer i entendre'l

Aquesta part està dividida en dues subparts. La primera és **Obrir l'aplicació.** En

aquesta part s'han de resoldre els possibles problemes que hagin pogut sorgir durant la primera execució de l'aplicació. La segona és **Realitzar captures.** Aquesta part comença quan ja puc obrir l'aplicació i el seu objectiu és realitzar la primera captura de tràfic d'internet. És possible que sorgeixin problemes, per tant caldrà resoldre'ls.

T3.Montar l'entorn de la nova aplicació

Aquesta part són els fonaments per poder començar a desenvolupar el projecte. Consisteix en instal·lar un tomcat, un eclipse, posar un jdk (java SE Runtime enviroment). Configurar el servidor de l'aplicació, tomcat, perquè utilitzi el port 80, la màquina virtual instal·lada i totes les llibreries necessàries per al funcionament de l'aplicació. Configurar l'ecilpse per utilitzar el framework de java "struts", per poder utilitzar ajax i per poder utilitzar hibernate en cas de que fos necessari accedir a alguna base de dades. Cal dir però, que aquestes són les feines imprescindibles per començar a desenvolupar el projecte, més endavant aquesta configuració es pot ampliar o podrà patir modificacions.

T4.Comunicació entre client (tomcat) i servidor de captures sense utilitzar cap ejb

En aquesta part es tracta de comprovar que es pot establir comunicació entre el client i el servidor de captures, realitzat amb c++ i utilitzant com a element fonamental la llibreria pcap. Cal dir que tot i poder patir alguna modificació el servidor de captures serà el mateix que el que utilitzava l'aplicació d'escriptori. En aquesta part s'estudiaran i s'utilitzaran les eines bàsiques, les quals s'utilitzaran més endavant a l'ejb, per a realitzar la connexió.

T5.Recuperar captures sense ejb

Un cop establerta la connexió entre client i servidor cal veure com es recuperaran les captures realitzades mitjançant el servei de corba d'esdeveniments. Bàsicament hauria de ser una classe java que recuperi les captures que retorna el servidor de captures. Les eines utilitzades serviran d'exemple més endavant, quan s'integri l'ejb.

T6.Montar ejb per realitzar les captures

Aquesta etapa consta de tres tasques clarament diferenciades. La primera d'elles és **Montar el servidor jboss en una màquina virtual diferent de la què utilitza tomcat**. Cal tenir en compte que s'haurà de configurar per a què treballi per ports diferents dels que utilitza tomcat i configurar el port que s'utilitzarà per a la comunicació entre jboss i tomcat. La segona tasca serà **Establir connexió entre jboss i tomcat**. Aquesta bàsicament consisteix en establir un pont de connexió entre

les dues màquines virtuals, mitjançant unes eines concretes, per un port concret configurat en el pas anterior. La tercera tasca consisteix en **Establir connexió entre jboss (ejb) i el servidor de captures.** En aquesta última tasca d'aquesta secció s'utilitzaran les eines ja estudiades en la tasca T4.

T7. Recuperar captures

Aquesta etapa consta de dues tasques clarament diferenciades. La primera d'elles és **Agafar les captures provinents del servidor de captures.** En aquesta tasca utilitzaré eines ja estudiades a la etapa T4. La segona part és **Utilitzar JDOM per crear un XML amb les captures.** Caldrà incorporar aquesta llibreria al projecte i aprendre a utilitzar-la.

T8. Mostrar les captures

Aquesta etapa consta de dues parts clarament diferenciades. La primera d'elles és **Accedir al XML de captures, mitjançant ajax, i montar una request les captures**. En aquesta tasca s'utilitzaran les eines de la llibreria JDOM per a poder treballar amb l'XML, s'utilitzarà ajax però sense utilitzar cap framework ja que resulta més senzill si tenim en compte la utilitat per la qual és necessari. Per acabar, a la request de la sessió se li passarà un objecte de tipus List amb totes les captures. La segona part és utilitzar la llibreria **Displaytag per mostrar les captures de la llista per pantalla**. Caldrà incorporar aquesta llibreria al projecte i aprendre a utilitzar-la.

T9. Implementar funcionalitats extra

Aquesta etapa consta de 5 tasques complementaries del projecte per tal de fer més còmode i utilitzable l'aplicació. La primera d'elles és **Guardar captures**. En aquesta tasca caldrà guardar l'arxiu XML de les captures que en aquell moment hi hagi a la pantalla. La segona és **Borrar captures**, és a dir, netejar la pantalla de les captures realitzades anteriorment. La tercera tasca és **Reconfigurar captures**, és a dir, poder canviar paràmetres com la ip, el port de la captura, el nombre de captures, etc. La quarta és **Obrir captures guardades** i la cinquena és **Refrescar la pantalla de captures** per si hi ha alguna captura que ha tardat més que les altres no s'havia mostrat per pantalla.

T10. Posar estils i estructura a l'aplicació

Per fer l'aplicació ordenada i vistosa caldrà triar i confeccionar uns estils i una estructura adequada.

T11. Mostrar altres sniffers de la xarxa.

En aquesta apartat cal fer una altre consulta al servidor de captures per mitja de l'ejb per tal de saber quins altres sniffers hi ha instal·lats a la xarxa.

T12.Detalls finals

En aquest apartat caldrà provar tota l'aplicació i comprovar que s'han assolit els objectius que es volien assolir en un principi. En cas de desemmascarar algun problema caldrà arreglar-l'ho.
En el següent diagrama de gantt es pot veure el pla temporal del projecte (Figura 13):

Pla Global:

Nom Tasca	Data d'inici	Data de final
T1.- Baixar sniffer i provar-l'ho	25/09/08	29/09/08
T2.- Fer funcionar l'sniffer i entendre'l	29/09/08	30/11/08
T2.1.- Obrir l'aplicació	29/09/08	20/10/08
T2.2.- Realitzar captures	21/10/08	30/11/08
T3.- Montar entorn de la nova aplicació	01/12/08	06/12/08
T4.- Comunicació entre client (Tomcat) i el servidor de captures sense utilitzar ejb	07/12/08	14/12/08
T5.- Recuperar captures sense ejb	15/12/08	20/12/08
T6.- Montar ejb 3.0 per realitzar captures	21/12/08	15/01/08
T6.1.- Montar jboss en una màquina virtual diferent de la de Tomcat	21/12/08	24/12/08
T6.2.- Establir connexió entre Tomcat i jboss	25/12/08	04/01/08
T6.3.- Establir connexió entre jboss i servidor de captures	05/01/09	15/01/09
T7.- Recuperar captures	16/01/09	28/01/09
T7.1.- Agafar captures provinents del servidor de captures	16/01/09	28/01/09
T7.2.- Utilitzar JDOM per crear XML amb captures.	19/01/09	28/01/09
T8.- Mostrar les captures	29/01/09	20/02/09
T8.1.- Accedir al XML mitjançant ajax i montar la request amb les captures	29/01/09	09/02/09
T8.2.- Utilitzar displaytag per mostrar les captures de forma dinàmica	10/02/09	20/02/09
T9.- Implementar funcionalitats extra	21/02/09	23/03/09
T9.1.- Guardar captures	21/02/09	27/02/09
T9.2.- Borrar captures	28/02/09	05/03/09
T9.3.- Reconfigurar captures	06/03/09	11/03/09
T9.4.- Obrir captures guardades	12/03/09	17/03/09
T9.5.-Refrescar pantalla de captures	18/03/09	23/03/09
T10.- Posar estils i estructura a l'aplicació	24/03/09	03/04/09
T11.- Mostrar altres sniffers de la xarxa	04/04/09	09/04/09
T12.- Detalls finals	10/04/09	20/04/09

Figura 13: Planificació temporal

4.5 Planificació econòmica.(Programador analista consultor)

L'estudi econòmic està enfocat a analitzar el cost del hardware utilitzat, del software necessari per a la implementació i documentació del projecte i, finalment, el cost en recursos humans.

4.5.1 Cost del hardware

Aquest cost correspon al cost de la màquina en la qual s'ha portat a terme el desenvolupament del projecte i un sistema d'alimentació ininterrompuda.

Per el desenvolupament s'ha utilitzat un ordinador de sobretaula amb les següents característiques:

-PC Intel Core 2 Quad a 3,1 Ghz
-4,0 GB de RAM
-750 GB HD
-Monitor de 20 pulgades
-Memòria Cache: 6 MB de cache de nivell 2
-Tipus de memòria RAM : DDR SDRAM
-Font d'alimentació de 270 W
-Placa base: P5K SE/EPU

El preu d'aquesta màquina és d'uns 1100 €.

Per al desenvolupament també s'ha utilitzat un Sistema d'alimentació ininterromput “riello ups iDialog plus”, el qual té un preu de 80 €.

Si a l'equip informàtic se li atribueix una vida útil de 4 anys i el projecte està planificat perquè duri uns 8 mesos, el cost proporcional sobre el hardware és de

16.67%((8/48)*100 =16.67), és a dir 197 € (1180*0.1667).

4.5.2 Cost del software

El cost corresponent al software utilitzat és 0 €, ja que tot el software utilitzat és software OpenSource.

Tot i això, per aconseguir el Linux Suse 11.0, que era la última versió del moment vaig haver de comprar una revista el cost de la qual era de 10 €.

4.5.3 Cost de recursos humans

Per veure els costs humans del projecte, hem dividit la feina amb quatre perfils diferents, ja que segons la tasca de portada a terme per cada perfil el cost per hora és major o menor. A continuació es detalla què realitza cada perfil:

Jefe de projecte: Encarregat de què el projecte compleixi els requisits sol·licitats, de realitzar reunions periòdiques (Punts de control) i responsable de complir els terminis pactats.

Analista: La seva funció és fer l'anàlisi de requeriments, especificació i disseny del sistema.

Programador: Realitza la implementació del sistema.

PERFIL	HORAS	PREU(€/hora)	SUBTOTAL (€)
Jefe del projecte	60	60	3600
Analista	300	30	9000
Programador	800	20	16000
TOTAL	1160		28600

Figura 14: Taula de planificació de costs del projecte

En la taula (Figura 14) no es reflecteixen altres costs, com podria ser la preparació del projecte, la elaboració de la memòria, preparació de la presentació, entre altres.

4.5.4 Cost total

Sumant tots els costs, la elaboració del projecte puja a 28806 €. Els costs es poden veure desglossats a la següent taula (Figura 15).

TIPUS COST	QUANTITAT (€)
Hardware	196
Software	10
Recursos Humans	28600
TOTAL	28806

Figura 15: Taula de planificació de costs totals

5 Disseny

5.1 Consideracions de disseny

El projecte ha consistit en transformar una aplicació d'escriptori en una aplicació web. Aquesta feina s'ha realitzat amb l'objectiu de millorar l'aplicació que hi havia feta.

Aquest canvi dóna a l'aplicació un gran poder d'utilitzabilitat per part de l'usuari final, ja que aquest no ha d'estar físicament davant de l'ordinador en el qual vol realitzar les captures de tràfic. Aquest pot accedir a través d'internet a l'aplicació web des d'un altre ordinador i realitzar les captures. De fet si suposem una xarxa de 5 ordinadors es podrien realitzar captures a tots els 5 ordinadors al mateix temps des de un sol ordinador, accedint a l'aplicació Sniffer dels altres 4 ordinadors.

A més a més en una aplicació d'escriptori normalment no iniciem una sessió per a cada utilitat de l'aplicació. S'inicia una sola sessió quant obrim el Sistema Operatiu. A l'aplicació d'escriptori l'usuari carrega l'sniffer i quan es fa la captura el programa realitza una crida al servidor de captures i aquest retorna al client la informació que es mostra per pantalla. A l'aplicació web és forca diferent. L'usuari posa la URL de l'aplicació en el navegador. Quan l'usuari realitza la captura, el navegador, per darrera, envia una request (Sol·licitud) al servidor web Tomcat utilitzant comunicació HTTP . Tomcat rep la request i sol·licita la captura a JBOSS(l'altre servidor de l'aplicació web) i aquest al servidor en C++ , que és el que realment ensuma(esnifa) la xarxa. Cal tenir en compte que JBOSS i Tomcat corren sobre Java Virtual Machines diferents. Per tant mentre es realitzen les captures la JVM de tomcat queda lliure. Aquest és un gran avantatge ja que l'aplicació pot continuar executant altres tasques sense necessitat d'haver de crear un threat. Mentre JBOSS rep les captures les va processant i guardant en un XML que tomcat anirà mirant a l'espera de captures.

El fet d'utilitzar dues JVM és molt important ja que d'aquesta manera la realització de les captures ni compromet ni bloqueja ni sobrecarrega l'aplicació web que corre a Tomcat. Això es transforma en més rapidesa per a l'usuari alhora de realitzar captures. És per tant, una aplicació formada per tres parts, les quals funcionen independentment una de l'altre. No es comprometen unes amb les altres i en el cas de que una quedi col·lapsada no compromet les altres i per tant per restaurar l'aplicació tan sols caldria posar en funcionament, rellançar, aquesta part i no tota l'aplicació.

La utilització d' AJAX dins de l'aplicació també busca un objectiu. AJAX permet fer consultes al servidor sense tenir que recarregar tota la pàgina. Aquest fet es

transforma en més rapidesa per a l'aplicació i sobretot dóna a l'usuari una bona impressió de l'aplicació.

Expliquem aquestes afirmacions:

- Per una banda, imaginem que volem realitzar 50 captures. I cada cop que ens arriba una captura nova l'aplicació ha de recarregar les captures que ja està mostrant pel navegador més la nova. Això significa una pèrdua de temps molt gran.

- Per l'altre banda, imaginem que l'usuari veu com el navegador recarrega la JSP 50 cops en un període curt de temps i sense ell haver pres cap més acció a l'aplicació que la de realitzar les captures. El funcionament normal a una aplicació web és que una JSP es carregui si l'usuari a premut una acció de l'aplicació, per tant, no donaria gaire bona impressió.

Com he comentat anteriorment, JBOSS guarda les captures que rep del servidor amb C++ a un fitxer XML. Això facilita a Tomcat poder agafar les captures de manera metòdica alhora que borra les captures que ja ha llegit. Un gran avantatge, per a l'usuari, és poder guardar aquestes captures en un fitxer XML amb la mateixa estructura que el generat per JBOSS per poder-les consultar més endavant. Per tant, utilitzant el mateix codi, podem tant obrir un fitxer XML de captures guardades per l'usuari, com anar llegint del fitxer XML a on JBOSS guarda les captures.

A la aplicació s'utilitzen “divs hidden”. DIV és un element html que s'utilitza per donar estructura. Per tant un div hidden és un element amagat. A l'aplicació s'utilitza per donar facilitats a l'usuari alhora de realitzar diferents accions com guardar la captura, obrir una captura o configurar una nova captura. En el moment en què l'usuari fa un clic a alguna d'aquestes accions el div passa a no ser un div amagat per passar a ser un div visible. Això facilita la feina de l'usuari ja que, per exemple, quan vol guardar una captura no es carrega una JSP diferent a la de captures; simplement el div es fa visible perquè pugui guardar les captures. Un cop guardades el div torna a quedar amagat. Aquestes accions s'havien realitzat, inicialment, com a “popups” però al final es va veure la utilització d'un div ocult era més adequat per a satisfer els requisits de l'usuari.

L'aplicació també té una acció que és borrar les captures de la pantalla. Imaginem que configures una captura i realitzes 10 captures. Si l'usuari vol realitzar més captures, tant sigui amb una nova configuració o amb la que ja tenia, les noves

captures es sumaran a les que ja tenia. Així no perd la informació que ja tenia i pot

guardar el conjunt de totes les captures en un únic XML. En el cas de voler, només, les captures que es realitzin en un cert moment i tenir ja captures a la pantalla del navegador d'altres captures, cal borrar-les prèviament. Aquest fet dóna, doncs, forca llibertat a l'usuari alhora d'organitzar les captures que va realitzant.

Un dels requisits de l'usuari és poder veure quants sniffers es troben a la xarxa i poder accedir a aquests. Per tant, a la part inferior esquerra de l'aplicació apareixen tots els sniffers que es troben a la xarxa. Aquests són linkables i ens envien directament a l'sniffer seleccionat. A més a més no perdem la connexió amb l'sniffer en el què ens trobàvem ja que aquest link obre una nova finestra al navegador enlloc d'accedir al nou sniffer a la mateixa finestra .

Finalment, cal recordar que el requisit més important que hauria de tenir l'aplicació seria poder realitzar captures. A l'aplicació les captures són configurables al principi, abans d'haver realitzat cap captura, amb una pantalla inicial però les captures es poden reconfigurar, per realitzar diferents tipus de captures, sense necessitat de tornar a aquesta pantalla inicial.

5.2 Legacy Code.

El codi del servidor de captures va ser extret d'una aplicació d'escriptori. Aquesta aplicació es diu sniffit i és opensource, i es troba a sourceforge.net.

El servidor de captures està implementat en c++. De tot el codi del servidor el més important per a la meva aplicació és el que es troba dins de la carpeta sniffer. El qual és el que realitza les captures. Les captures les realitza amb l'ajuda d'una llibreria “pcap” la qual és molt coneguda i és la base de molts sniffers. PCAP és una llibreria que només funciona a linux i està feta per codi en c++. Per aquest motiu el servidor de captures està implementat en c++.

La part més important i interessant és el punt en el qual es prepara i es llenca la captura. El mètode “launch” és el que llenca la captura. Aquest és cridat per “pthread_create”. Per tant veiem com el servidor crea un fill d'execució per llencar la captura. Els paquets de les captures són parsejats i retornats per el mètode “packetHandler”. Aquest mètode es passa a la funció de pcaplib “pcap_loop”.

Abans de començar la captura és important **buscar el dispositiu** (device) i obtenir la **direcció de xarxa** i la seva **mascara.** La llibreria pcap ens dóna les eines necessàries per portar-ho a terme: “pcap_lookupdev” i “pcap_lookupnet”. Finalment tan sols

queda començar la captura amb “pcap_open_live” i “pcap_loop”.

Per a fer funcionar l'aplicació en el meu pc vaig tenir problemes precisament amb la inicialització de l'entorn per realitzar la captura. Aquest i altres temes els trobem comentats en el següent apartat.

5.3 Canvis a legacy code.

En aquest apartat parlaré dels canvis que vaig fer a al servidor per a què funcionés tal i com necessita l'aplicació web.
Dels primers canvis va ser modificar el fitxer Makefile.am ja que el sistema de compilació automake a canviat el seu comportament respecte de la versió utilitzada inicialment. Aquests fitxer serveix per generar el dimoni de l'sniffer. El makefile.am ja creava un binari anomenat snifferd però jo vaig afegir la creació d'un altre binari sobre el qual fer proves de compilació. Això va ser degut a què no aconseguia compilar i generar el binari amb la configuració que hi havia. Finalment, a la compilació del nou binari anomenat “capred” vaig afegir les llibreries -lncurses -lpcap i -lssl.

Pel què fa al codi he comentat totes les comprovacions d'usuari ja que a l'aplicació no seran necessàries. En el futur s'instal·larà un sistema de login a l'aplicació web i per tant no serà necessari les comprovacions a dins del servidor de captures. Una bona opció seria incorporar algun sistema com “josso” . Josso és un projecte que aniria apart del war de l'aplicació i correria també dins del tomcat. Amb “josso” es poden definir quines URL's es volen controlar i quins permisos i privilegis dones als usuaris que entren dins de la regió controlada.

Per a què funcionés l'aplicació vaig haver de canviar la inicialització de l'entorn per a la captura ja que la funció, “launch”, encarregada de llençar la captura, es quedava penjada. La inicialització de l'entorn, que es passava per paràmetre, la vaig posar just abans de llençar la captura amb “pcap_open_live”, a dins de la funció “launch”. Per descobrir el què estava fallan van ser molt útils les funcions de la llibreria pcap “pcap_perror” a la qual li passes el “device” i la descripció de la direcció de xarxa i la seva mascara i t'informa de possibles problemes. Existeix una altre funció útil “pcap_geterr” a la qual només li passes la direcció de xarxa i la seva mascara.

La funció “checkID” va ser comentada en un principi tot i que al final s'ha descomentat. Es tracta d'un ID per diferenciar captures. Aquest es treu de la funció autogenerada “ByteArrayHolder”. Aquesta fa una consulta al servidor i aquest li retorna una array de bytes. Quan es realitza la captura es comprova, es fa un check, d'aquest array per saber si és el que ha proporcionat anteriorment el servidor. Aquest check consisteix en desencriptar la marca. Es va canviar el que retornava el servidor per “byte[] returnData={0};”, és a dir, per un id vuit.

Per a cada dimoni s'inclou una clau privada i un certificat per a poder autenticar l'usuari. Aquestes no es volen utilitzar a la nova aplicació. No és necessari utilitzar aquest sistema per validar l'usuari tal i com hem comentat anteriorment. Per tant es va comentar el codi que hi té relació. A l'aplicació d'escriptori, en el client, es crea un password d'usuari generat amb el certificat i s'envia al servidor. Un cop al servidor es fa una comprovació, validació, d'aquest password amb la clau privada. A la nova aplicació, des del client no es genera el password i des del servidor no es fan comprovacions del password. Per tant ha fet falta treure, també totes les excepcions relacionades amb aquestes comprovacions.

Totes les referències a la clau pública o privada com pot ésser per exemple : "getPublicKey(PUBLIC_KEY)" han estat també comentades. El codi comentat es troba bàsicament a dins de les carpetes server-beta/daemons/common i server-beta/daemons/sniffer.

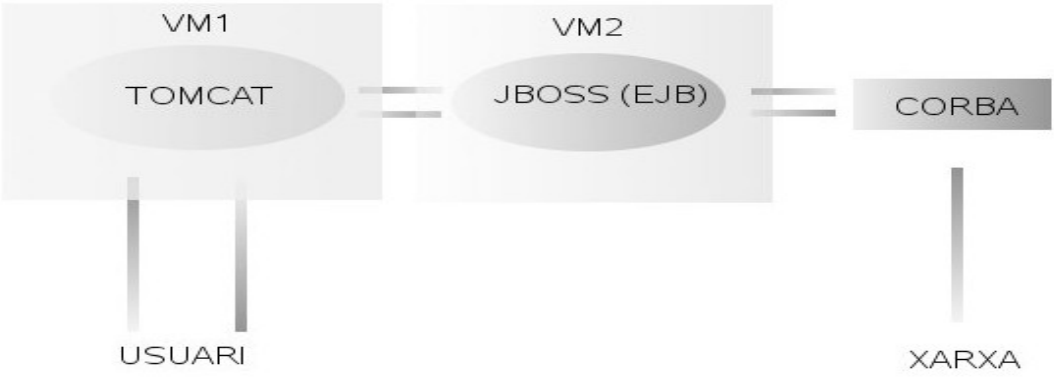
S'utilitzen algunes funcions de la llibreria "pcap" dins del codi que no estan controlades en cas d'error. He afegit codi per tal de què es llenci una excepció "runtime" en cas d'una mala execució.

5.4 Aplicació web.

Aquesta apartat és molt important per a futurs millores o ampliacions del projecte ja que explicaré com funciona l'aplicació per dins.

Primer de tot donaré una visió global de l'estructuració del projecte. A l'aplicació podem diferenciar 5 parts molt importants; l'usuari o client de l'aplicació, el servidor de l'aplicació TOMCAT, el servidor de l'ejb JBOSS, l'aplicació CORBA(Servidor de les captures) i la xarxa. A la Figura 16 es pot veure un esquema de tota l'aplicació en el qual podem veure la interconnexió entre les parts més significatives d'aquest.

Figura 16: Esquema general de l'aplicació



5.4.1 Parts aprofitament de l'aplicació d'escriptori

Quan vaig començar a desenvolupar l'aplicació un dels principals objectius va ser aprofitar al màxim el codi del client de l'aplicació d'escriptori. Així va ser, un cop sabia com s'estructurava i funcionava l'aplicació d'escriptori va ser relativament fàcil aprofitar al màxim certes parts del codi ja teclejat de l'antic client.

Bàsicament vaig aprofitar les parts que s'encarreguen de la inicialització i obtenir l'ORB i el POA(RootPOA), la part que s'encarrega d'agafar el Servei de Noms, la part que agafa el servei d'esdeveniments, la part per inicialitzar els dimonis, la part que s'encarrega de registrar les diferents llibreries necessàries per a l'aplicació. Aquest codi es troba a l'aplicació d'escriptori dins de "corba.java". Cal dir, però, que el codi va patir en algun cas petites modificacions. Per exemple, de tots els dimonis que tenia l'aplicació d'escriptori tan sols m'interessava la part de l'sniffer, per tant vaig, en certa manera, tot i aprofitar el codi de l'antic client, aquest, va ser reestructurat i modificat.

Una altre part la qual va ser aprofitar van ser les diferents consultes que l'ejb fa al servidor de captures. La part de consultar quants sniffers conviuen a la xarxa i la part de la consulta de captures. A la consulta de les captures alhora de construir el filtre també es va aprofitar forca el codi.

La part de PushConsumerCaptures, encarregada de recollir les captures provinents del servidor de captures, tot i que el tractament de les captures es realitza diferent, es va aprofitar parcialment ja que en aquest moment es precisament quant es crea l'xml de captures.

També vaig aprofitar alguna excepció de codi anterior com per exemple la daemonexception, la eventexception, la namingexception i la POAexception. Totes elles estenen de "java.lang.Exception".

5.4.2 El codi de Tomcat

El codi que s'executa dins del servidor tomcat. Aquest és l'encarregat de gestionar totes les peticions de l'usuari. Aquesta part és bàsicament una capa de negoci.

El llenguatge de programació d'aquesta part és JAVA. Aquesta part del codi ha estat desenvolupada amb l'ajuda d'eclipse. Eclipse és bàsicament un entorn integrat de codi obert (OpenSource) multiplataforma per desenvolupar aplicacions.

Per al codi en JAVA s'ha utilitzat el framework d' Struts per agilitzar el desenvolupament del projecte. Struts és una eina de suport per al desenvolupament

d'aplicacions web sota el patró MVC (Model Vista Controlador)(Figura 17) per a la plataforma J2EE.

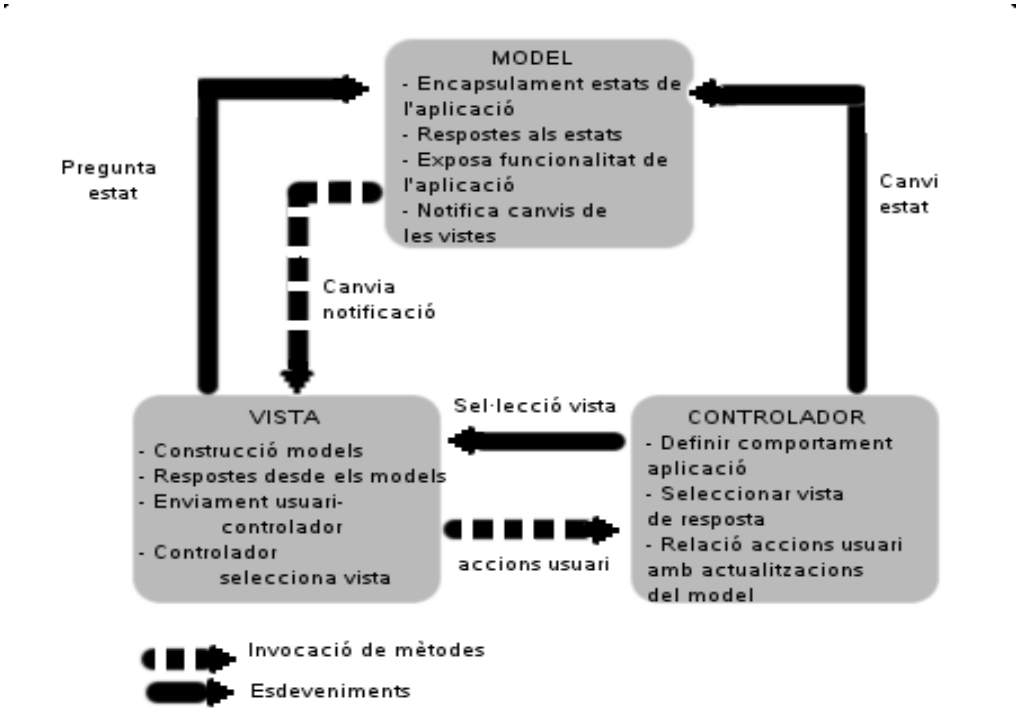


Figura 17: Esquema MVC

Desenvolupar una aplicació web sense utilitzar cap framework és molt costs. La utilització d'struts a part fa que qualsevol persona que sàpiga com està estructurat aquest framework sigui capaç d'entendre el codi font.
L'aplicació no utilitza base de dades, però està preparada per poder-ne utilitzar gràcies a MVC i a què si ha configurat hibernate.

5.4.2.1 Estructura

A dins del projecte trobem dues parts molt importants: la carpeta **src** i la carpeta **webcontent**.

A)SRC

Dins de la carpeta **src** del projecte tenim les següents carpetes i arxius:

- *daemons*
- *daemons.events*
- *daemons.ObjectFactoryPackage*
- *daemons.SecureObjectPackage*

- *header*
- *main.java.com.uni.sniffer.actions*
- *main.java.com.uni.sniffer.exceptions*
- *main.java.com.uni.sniffer.forms*
- *main.java.com.uni.sniffer.model*
- *main.java.com.uni.sniffer.parameters*
- *main.java.com.uni.sniffer.service*
- *packet*
- *libraries*
- *build.xml*
- *hibernate.cfg.xml*

Primerament, explicaré les carpetes les quals contenen codi auto generat per el compilador de l'OpenORB per tal poder establir per l'ORB

En el header: hi ha desenvolupades diferents funcions. Quasi tots els arxius són auto generats per el compilador OpenORB. A partir de l'idl el compilador OpenORB crea tres fitxers per el client: un stub, un helper i un holder. Aquest arxius ens permeten tenir el pont de comunicació ORB. Cada una d'aquestes funcions conté la seva interfície (classes abstractes) amb la declaració de mètodes i els seus mètodes Helper i Holder d'OpenORB.

La classe Helper serveix per a fer referència a un objecte remot i recuperar-ne un valor. A més a més el mètode Helper assegura que l'objecte està encastat correctament. És a dir, que la classe a la qual volem assignar l'objecte coincideix amb la classe que estem recuperaran.

La classe Holder conté els valors retornats quan s'han utilitzat paràmetres inout o out a un mètode.

Per a la comunicació entre client i servidor es necessiten uns adaptadors, la funció dels quals sigui transformar una invocació local a una petició ORB. A la part del client es diuen stubs i a la part del servidor es diuen skeletons.

També trobem la implementació de l'interfície i la seva DefaultFactory, els quals no són generats a partir de la compilació de l'idl. La funció de la Factoria és que donat un objecte del tipus “org.omg.CORBA_2_3.portable.InputStream” ens retorna un objecte el qual és la implementació de la clase abstracta bàsica. Per tant aquest objecte implementació, bàsicament, processa les dades, informació, que arriben per l'objecte “org.omg.CORBA_2_3.portable.InputStream”, a través del pont ORB. També podem dir que mitjançant les factories es creen nous objectes per facilitar

l'accés als diferents components.

A la carpeta **daemons.events:** Hi trobem tres classes java generades per el compilador OpenORB. La classe Pending i les seves respectives classes Helper i Holder. Aquestes ens ajudaran a agafar l'esdeveniment que vingui per el canal d'esdeveniments de l'ORB. De fet s'utilitzarà en el moment de recuperar les captures.

A la carpeta **daemons.ObjectFactoryPackage:** Trobem els arxius generats, per el compilador OpenORB, per a dues possibles excepcions: BadOperation i BadUserOrPassword. De fet BadUserOrPassword no s'utilitza a l'aplicació actual.

A la carpeta **daemons.SecureObjectPackage:** Trobem els arxius generats, per el Compilador OpenORB, per a dues possibles excepcions: BadOperation i BadMark.

A la carpeta **packet:** Trobem les classes generades per el compilador OpenORB que tenen referència amb el paquet de captures. PacketType, TimeMark i PacketArray. Aquesta última classe no té classe abstracta ja que no conté variables. Cal dir també, que la classe PacketType té una implementació i una factoria per processar millor les dades de les captures.

A la carpeta **daemons:** Trobem els diferents Stubs per a cada un dels IDL's compilats.

També trobem els arxius generats per el compilador OpenORB per a diferents dimonis: Capture, Pumper, Empty, Factory, Sniffer, Source, Filtre, UdpSource, UdpSink, TcpSource, TcpSink, ObjectFactory amb les seves respectives classes POA (Portable Object Adapter) per a l'assignació de recursos i amb les seves classes Helper i Holder.
POA millora l'antic BOA (Basic Object Adapter). Tot el comportament del POA es controla mitjançant polítiques establertes quan es crea l'adaptador i no poden ésser canviades durant la vida d'aquest. El ORB proporciona un “POA root” amb les polítiques estàndards a partir de les quals s'han creat els POA d'aquesta carpeta. La seva utilitat és l'assignació i alliberació de recursos mitjançant les polítiques POA. És a dir, determinen si les referències a objectes són transitòries o persistents.

Les altres carpetes contenen arxius propis del projecte sense arxius generats de manera automàtica per el compilador ORB:

A la carpeta **main.java.com.uni.sniffer.actions:** Hi trobem accions necessàries per al funcionament de l'aplicació. Aquestes classes java estenen de la classe action d'struts i són les classes que tramiten les peticions “URL.do” de l'usuari. Com per exemple l'acció “OpenCapturesAction” el qual és cridat quan l'usuari de l'aplicació obre alguna captura que tenia guardada.

A la carpeta **main.java.com.uni.sniffer.exceptions:** Trobem les excepcions que s'utilitzen en els actions.

A la carpeta **main.java.com.uni.sniffer.forms:** Trobem una llista dels formularis que arriben als “actions” provinents de les jsp's.

A la carpeta **main.java.com.uni.sniffer.model:** Trobem algunes classes java i algun mapping d'hibernate d 'exemple per si en el futur es volgués utilitzar base de dades.

A la carpeta **main.java.com.uni.sniffer.services:** Trobem serveis que utilitzen els actions.

A la carpeta **main.java.com.uni.sniffer.parameters:** Trobem una classe java anomenada constants, la qual conté constants de configuració de l'aplicació.

Trobem dos arxius molt importants **build.xml** i **hibernate.cfg.xml**. EL primer d'ells és necessari per a la compilació del projecte. El build.xml pot ser d'utilitat en un futur per si l'aplicació necessita ésser compilada de manera especial utilitzant “apache-ant” .Per exemple, per ajuntar més d'un projecte en el mateix war. El segon serveix per configurar els mappings d'hibernate. Ara, hi ha un mapping d'exemple.

Finalment, trobem una carpeta anomenada “Libraries” la qual conté totes les llibreries incloses en el buildpath del projecte.

B)WEBCONTENT

A dins aquesta carpeta trobem les següents carpetes i arxius:

- *css*
- *img*
- *js*
- *META-INF*
- *WEB-INF*
- *index.jsp*

A dins la carpeta **css:** Trobem els diferents css que s'utilitzen per donar estils i format a l'aplicació web.

A dins la carpeta **img:** Tal i com indica el seu nom, a dins es troben les imatges que utilitza l'aplicació.
A dins la carpeta **js:** Trobem un arxiu js, és a dir, codi JAVASCRIPT. El fitxer es diu ajax.js i hi ha funcions per poder fer la crida ajax que serveix per anar mostrant les

captures.

A dins la carpeta **META-INF:** Dins aquesta carpeta trobem la carpeta de maven trobem els arxius **pom.properties** i els **pom.xml.** Les llibreries de maven han d'anar configurades dins aquests arxius, en els quals entre altres coses s'indica la versió d'struts que s'utilitza. Fora de l'arxiu de maven trobem l'arxiu MANIFEST.MF el qual s'utilitza per executar jar's. En aquest cas és un fitxer autogenerat per struts durant la seva instal·lació.

A dins la carpeta **WEB-INF:** Dins aquesta carpeta trobem tres subcarpetes: la carpeta **classes** no té molta importància. Tot i això i trobem l'arxiu MessageResources.properties, dins el qual hi ha l'assignació d'una clau als literals que s'utilitzen a l'aplicació. Aquestes claus s'accedeixen a les jsp's. La carpeta **lib,** a on es troben les llibreries que s'utilitzen a l'aplicació. A diferència de la carpeta libraries, ubicada dins src, dins la carpeta lib només hi trobem les llibreries que es troben físicament dins del projecte i no totes les del BuildPath. La carpeta **pages** conté totes les jsp's que s'utilitzen a l'aplicació.

Fora d'aquestes carpetes trobem molts arxius: Els arxius **Displaytag.properties** i **Displaytag.tld** tenen a veure amb la llibreria Displaytag que s'utilitza per mostrar les captures, és a dir, que construeix la taula de captures. Els arxius **struts-bean.tld**, **struts-config.xml**, **struts-html.tld**, **struts-logic.tld** i **struts-nested.tld** tenen a veure amb el framework d'struts. Els arxius tld ens proporcionen funcions per facilitar la programació de les jsp's i l'arxiu xml conté la configuració de les accions, formularis d'aquestes i jsp's a les quals va dirigida l'acció. També trobem un arxiu **validation.xml** en el qual es poden configurar validacions de formularis. Aquest no s'utilitza a l'aplicació. Trobem el conegut arxiu **web.xml**, en aquest arxiu anirien configurats els servlets i filtres de l'aplicació però utilitzant struts no és necessari. Podríem dir què aquesta configuració es troba a struts-config.xml. De tota manera s'hi troba configurada la primera pàgina d'accés a l'aplicació, la llibreria Displaytag, i el mapping dels actions d'struts amb el fitxer que conté la informació, el ja anomenat struts-config.xml.

A fora d'aquestes carpetes trobem una jsp, anomenada **index.jsp**. Aquesta és la primera pàgina a la qual s'accedeix quan entrem per primer cop a l'aplicació. Aquesta tan sols conté una redirecció. Aquesta pàgina, utilitzant un tag d'struts, ens redirigeix a l'action welcome.do i aquest a la primera pàgina de l'aplicació amb contingut que és la pàgina welcome.jsp.

5.4.2.2 Interconnexió del codi

Un cop sabem que hi ha a cada carpeta anem a veure com s'entrellacen els diferents elements.

El primer que mostraré és un gràfic en el qual es pot veure que fa l'aplicació quan entra un usuari (Figura 18).

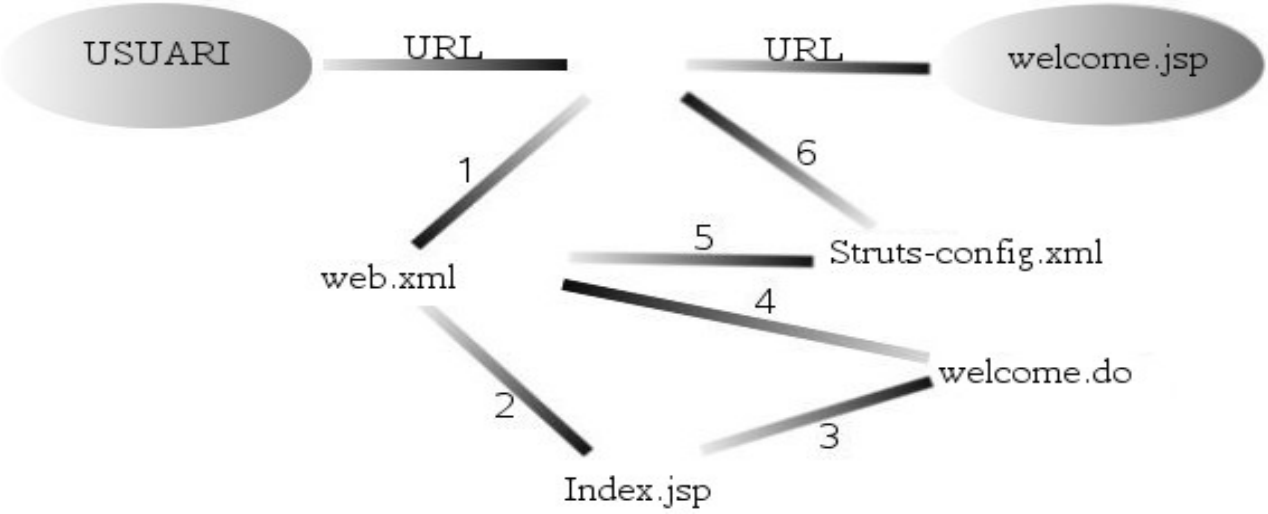


Figura 18: Esquema interconnexió codi amb struts.

L'usuari escriu la URL, com en tota aplicació web java es consulta el web.xml per saber quin és el welcome file. El web.xml redirigeix la consulta a index.jsp i aquest mitjançant un tag d'struts llança l'action welcome.do. Aquest action mira a web.xml el seu mapping com si es tractés d'un servlet, pas 4 . El web xml el redirigeix a struts-config.xml i aquest l'envia a welcome.jsp, la pàgina de benvinguda de l'aplicació. L'action welcome conté un formulari LoginForm, el qual no s'utilitza però en un futur podria utilitzar-se per fer una validació d'usuari.

Un cop hem explicat el què fa l'aplicació quan entra l'usuari. Explicarem amb l'ajuda de gràfics cada una de les accions que realitza l'aplicació.

La primera acció és la de sol·licitar **una captura**. En aquesta acció es posen en sessió les dades que configuren la captura i es consulta al servidor de captures, per mitja de l'ejb, tots els sniffers que coexisteixen a la xarxa. Podem veure el recorregut de l'acció en el següent gràfic (Figura 20).

L'usuari clica el botó per sol·licitar la captura de la pàgina welcome.jsp (Figura 19). Es llença l'acció SolicitaCapture.do aquesta, com si es tractés d'un simple servlet mira el seu mapping al web.xml. El Web.xml l'envia al fitxer struts-config.xml. L'struts-config indica quin és el codi java que s'ha d'executar i quina és la jsp a la qual ha d'anar. En el codi JAVA, després del pas 4, s'agafa els valors de configuració de les captures i es posen en sessió . També es posen en sessió els sniffers que coexisteixen a la xarxa després d'haver consultat al servidor de captures quins són. Un cop hem posat els elements a la sessió i si no hi ha hagut cap error l'acció ens condueix a la jsp ShowCapture.

El fet de posar les dades a la sessió fa que l'usuari no hagi d'entrar les dades en el moment que vol fer la captura. Les dades queden guardades mentre la sessió no caduqui i l'usuari pot iniciar la captura més endavant. Aquesta dades tan sols poden ésser canviades si l'usuari entra a l'apartat de configuració. La figura següent mostra la pantalla de configuració inicial (Figura 19)

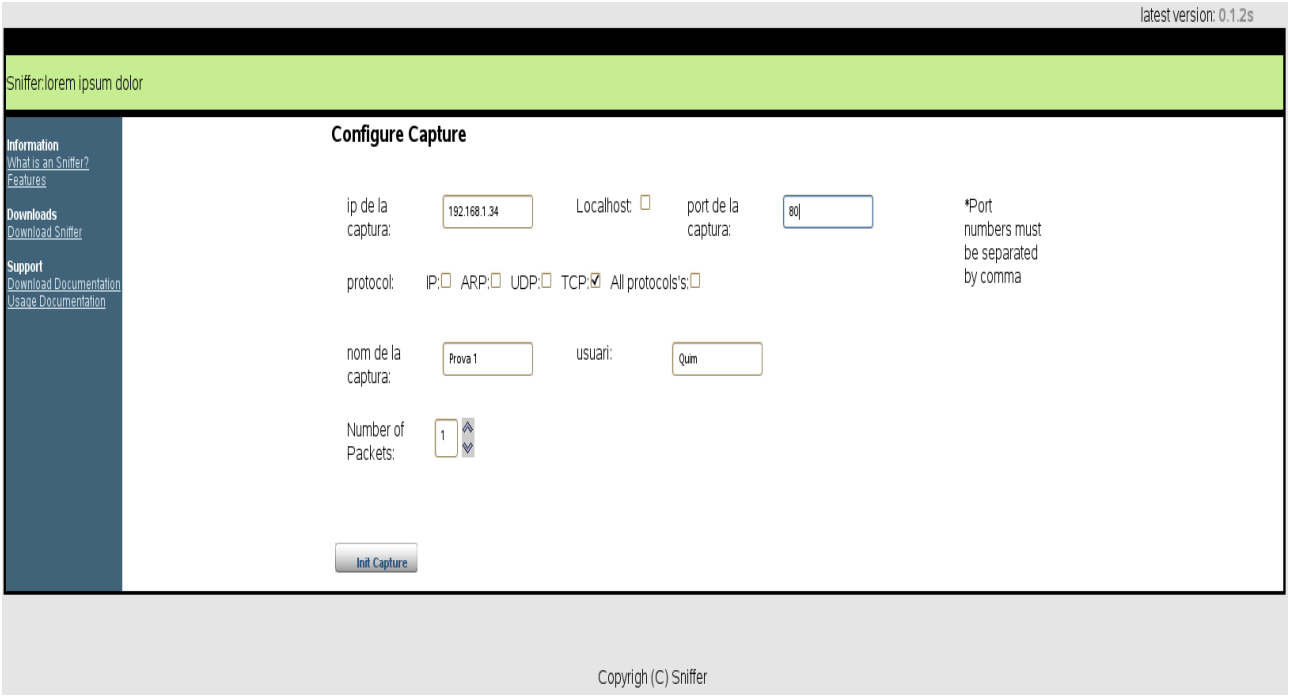
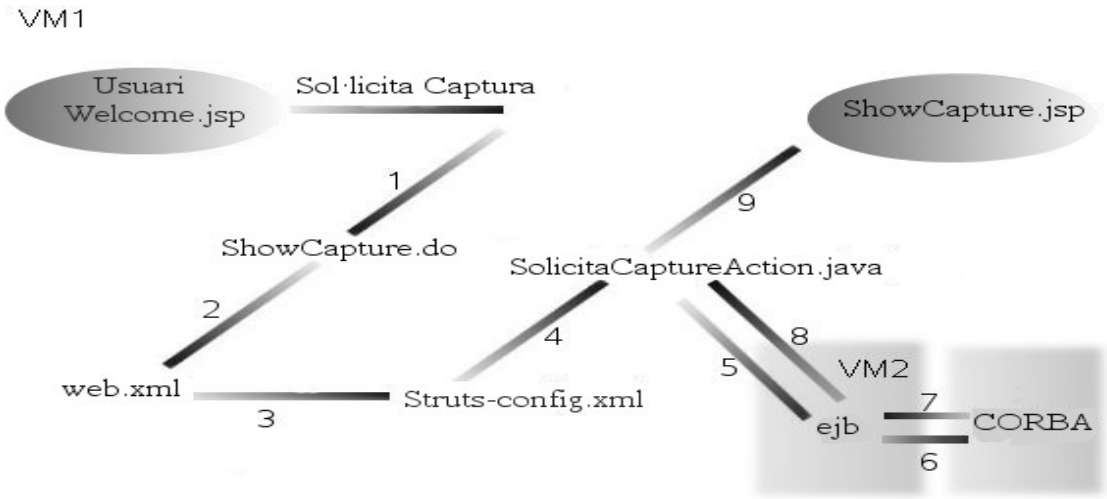


Figura 19: Pantalla Inicial

Figura 20: Crida de captures



L'acció la qual podríem dir que és la més important és la de realitzar les captures i mostrar-les per la pantalla. (A la Figura 20 es pot veure què fa l'aplicació per dins)

L'usuari es troba a dins de l'aplicació un cop ha introduït les dades per configurar la captura (Figura 22) i clica el botó per començar la captura. Es crida l'acció PresentCapture.do aquesta agafa les dades de configuració de la captura que es trobaven en sessió i crea un threat, el qual serà l'encarregat de cridar a l'ejb perquè s'encarregui d'enviar les dades configurades al servidor de captures i de indicar-l'hi que pot començar la captura. Aquest action tan sols s'encarrega de dir a l'ejb que comenci la captura. L'ejb farà la consulta al servidor de captures, utilitzant eines de corba, i s'encarregarà d'omplir un fitxer xml amb les captures realitzades. Un cop a la jsp final, ShowCaptures.jsp, aquesta, mitjançant ajax, farà consultes a l'acció CheckForCaptures.do per agafar les captures que l'ejb va deixant a l'xml. L'acció CheckForCaptures agafa les captures de l'XML les processa i crea una llista d'objectes "Capture". Un cop a la jsp amb l'ajuda de la llibreria Displaytag es crea de forma dinàmica la taula amb les captures.

En aquesta tasca trobem dos fets que cal explicar:

La utilització d'un **ejb** és important ja que així separem la part que crida al servidor de captures i en rep la seva informació de l'aplicació en si. Aquestes dues parts es

troben en màquines virtuals diferents. Això fa que el sistema de l'aplicació no es estigui afectat per l'altre part.
La utilització d'un **threat** per cridar l'ejb és important ja que així el fill principal pot continuar la seva tasca i carregar la pàgina ShowCaptures.jsp.
La utilització d'un **xml** és també important per separar, és a dir, fer independents l'ejb de l'aplicació. Per una banda l'ejb va posant captures dins l'XML i l'aplicació les va agafant. Podríem dir que l'ejb seria el productor i l'aplicació el consumidor ja que aquest va esborrant, de l'xml, les captures què agafa.
La utilització d'**Ajax**, com ja s'ha comentat anteriorment, és important perquè l'usuari de l'aplicació tingui de rapidesa, ja que no es recarrega la pàgina per cada captura que s'agafa.

La utilització de la llibreria **DisplayTag** facilita poder tenir una taula de captures dinàmica sense haver de posar a la jsp molta lògica.

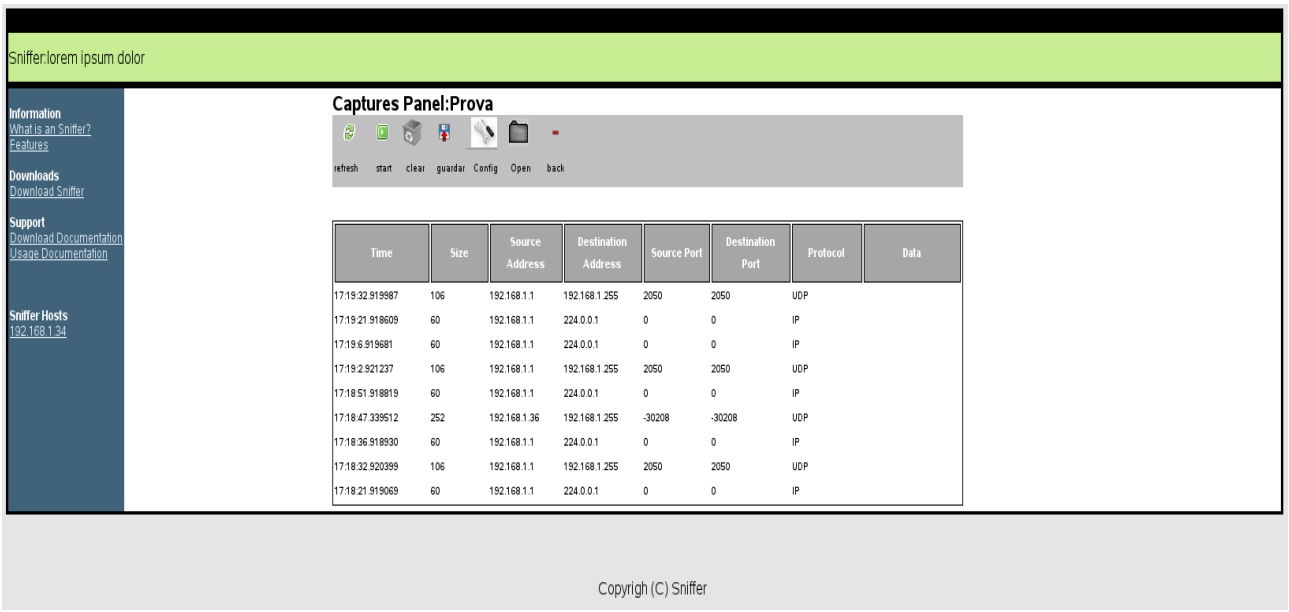


Figura 21: Llista de captures amb DisplayTag

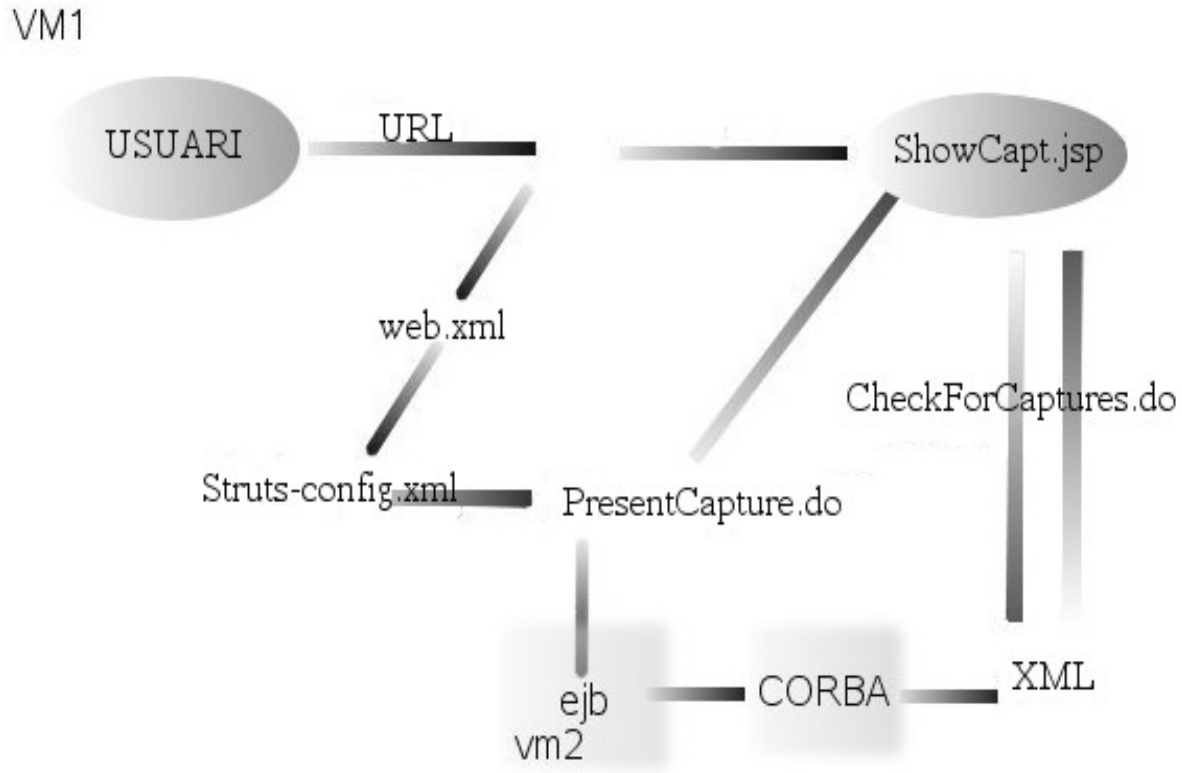


Figura 22: Recuperació de les captures

Una altra acció important és la de guardar o obrir una captura. Els actions que s'encarreguen d'aquesta acció són SaveCaptures.do i OpenCaptures.do.

L'usuari fa un clic a l'opció del menú “save” i s'obre un div (Es pot veure a la Figura 23), que es trobava amagat, per poder guardar les captures. L'usuari diu a on el vol guardar i clica el botó “save”. El web.xml redirigeix el .do cap al fitxer struts-config.xml i aquest al codi java de l'acció “SaveCapturesAction” i, un cop guardat, retorna a la pàgina ShowCaptures.jsp. Per obrir una captura l'usuari clica a l'opció del menú “obrir” i, igual que abans, s'obre un div que estava ocult (Es pot veure a la Figura 24), quan l'usuari ha escollit el fitxer a obrir i clica al botó obrir el web.xml redirigeix el .do cap al fitxer struts-config.xml i aquest al codi java OpenCapturesAction. Aquest retorna a la jsp una llista de captures perquè siguin mostrades.

A la pantalla de guardar captures es pot introduir la ruta a on vols guardar les

captures. Tot i això l'action guardarà també l'xml de les captures, a una ruta concreta, dins el servidor. Alhora d'obrir captures es pot navegar per l'ordinador i escollir un xml de captures, però aquest només serà obert si prèviament a estat guardat per

l'aplicació. L'action d'obrir captura només busca dins del servidor tot i que l'usuari no

en tingui percepció. Aquest comportament es deu a què el tag html "<input type='file'" quan arriba a l'action, a través del formulari, des de la pantalla d'obrir captures envia la ruta relativa de l'xml i no la ruta absoluta. Aquest fet és inherent al tag per motius de seguretat. Per aquest motiu en el moment de guardar l'xml es guarda en una ruta coneguda dins el servidor.

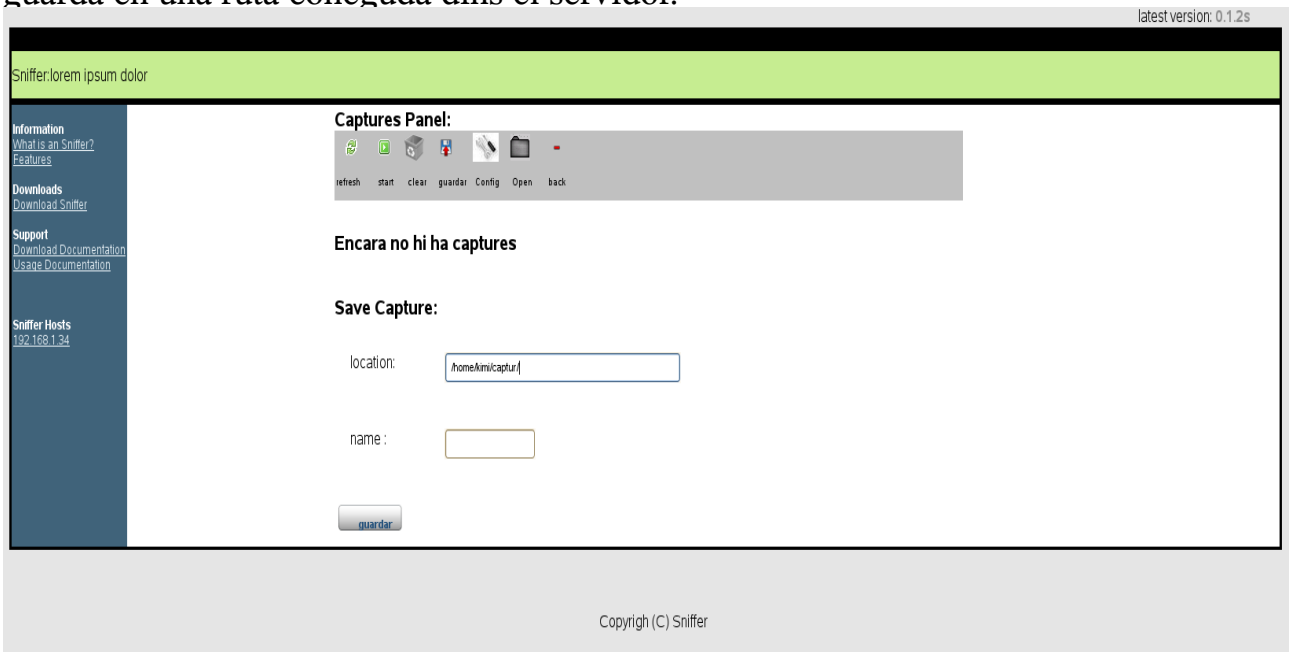


Figura 23: Pantalla de captures

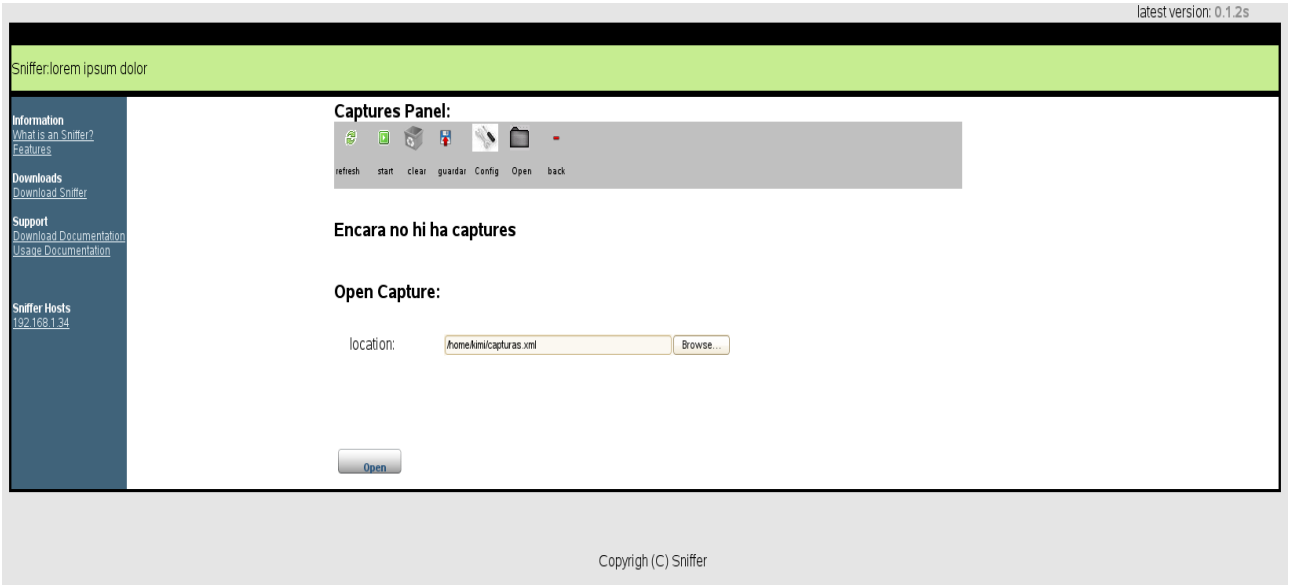


Figura 24: Obrir captures

Una altre opció molt interessant que té l'aplicació és la de reconfigurar la captura. Tal

i com passa amb les altres funcionalitats, l'usuari escull l'opció “config” i s'obre un div que es trobava ocult (Es pot veure a la Figura 25). L'usuari omple les noves dades per a la captura i clica el botó de sota per confirmar-ho. El web.xml redirigeix el .do cap al fitxer struts-config.xml i aquest al codi java de l'acció “SolicitaCapturesAction”. Aquest action posarà les noves dades de la configuració de captura a la sessió. L'action que crida és el mateix que el que s'utilitza per a la primera configuració, però una variable que es passa com a paràmetre li diu que es tracta d'una segona configuració i que per tant no ha de fer la consulta per saber els sniffers que hi ha a la xarxa. Aquests ja es troben guardats a la sessió.

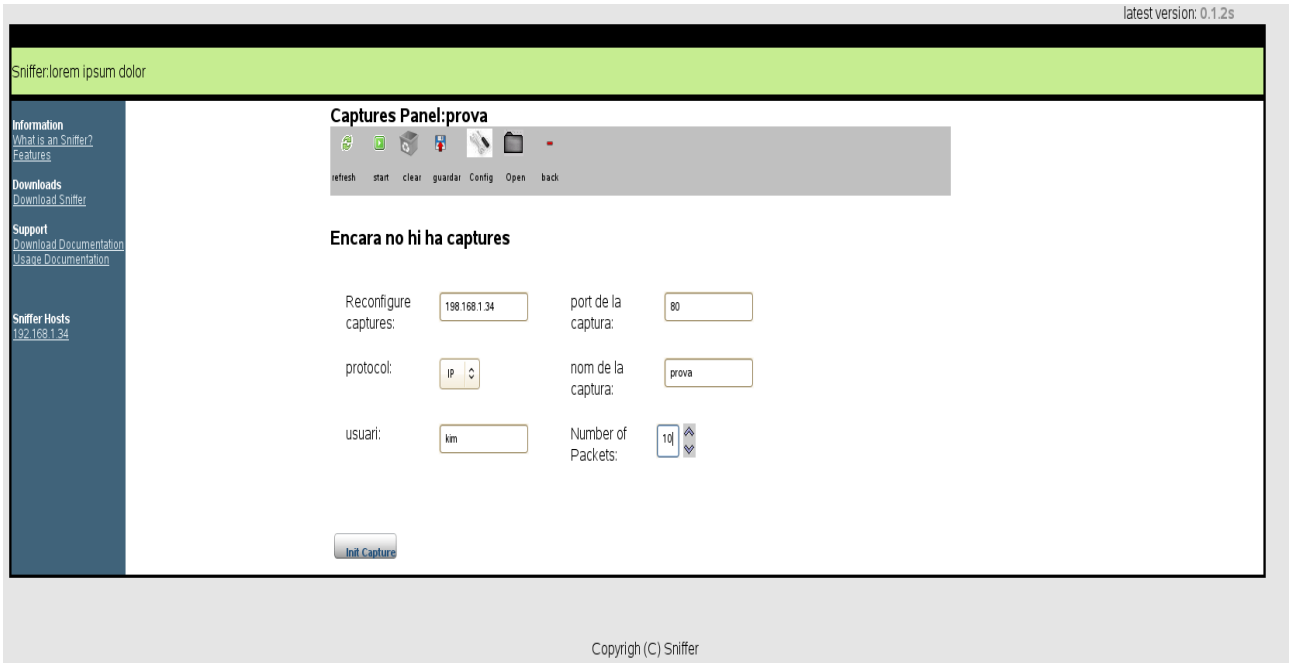


Figura 25: Reconfigurar captures

Podem també fer un “**refresh**”. Quan realitzem una captura la jsp, per mitjà d'ajax, accedeix al servidor per recuperar les captures. En total fa 10 crides ajax espaiades entre 1 i 3 segons. Tot i això hi ha captures que tarden a arribar i podria ser que s'escapessin al control. Per aquest motiu l'usuari sempre pot refrescar la pàgina per assegurar-se que no hi ha cap captura que no hagi estat mostrada. Aquesta acció simplement fa una consulta al servidor per veure si hi ha alguna captura i recarrega la pàgina.

També trobem l'acció d'**esborrar les captures** realitzades. Aquesta acció simplement esborra de la sessió l'objecte que conté la llista de les captures realitzades i refresca la pàgina per tal que el displaytag actualitzi el llistat de captures.

També tenim l'opció d'anar endarrere. Aquesta funcionalitat és útil si em realitzat alguna acció i volem veure les dades que teníem abans.

5.4.3 El codi de l'ejb

La funcionalitat de l'ejb és fer de passarel·la entre l'aplicació i el servidor corba de captures. Quan l'aplicació vol obtenir captures o quan vol saber quants sniffers hi ha a la xarxa; avisa a l'ejb i aquest és el que s'encarregarà de fer la consulta al servidor corba i gestionar la resposta.

Bàsicament, l'ejb esta compost per una interfície, CaptureBeanRemote, a on es defineixen els mètodes d'aquest i una classe CaptureBean que implementa la interfície remote.

L'ejb té dues funcions la primera és llençar una captura de tràfic d'internet i la segona és buscar tots els sniffers que hi ha a la xarxa:

Trobar els sniffers que conviuen a la xarxa

L'ejb rep la petició per el port 1099 i la classe captureBean la processa. Aquesta funcionalitat és sempre la primera que s'utilitza. Per tant és l'encarregada d'iniciar l'ORB, el servei de noms, el canal d'esdeveniments, els dimonis i de registrar les factories i finalment d'arrencar l'ORB. Simplement fa una consulta al servidor de captures.

Llençar una captura de tràfic d'internet

L'ejb rep la petició de l'aplicació que corre sobre tomcat per el port 1099. Aquesta petició serà processada per el mètode “StartCaptureBean” de la classe “captureBean.java”(El procés per començar una captura es veu a la Figura 26). L'ejb s'encarrega de començar la captura i inicialitzar tots els elements necessaris com el filtre de la captura i el consumidor. El consumidor “CapturePushConsumer” extent de la classe pushConsumer de CORBA. El consumidor conté el mètode push el qual rebrà els paquets de la captura. Els processa i els guarda dins un fitxer xml. (El procés del consumidor es pot veure a la Figura 27).

El codi necessari per la comunicació a través de l'ORB es trobava a dins de l'aplicació que corre amb tomcat ja que en un primer moment no es tenia pensat l'utilització d'un ejb. Més endavant es va veure que era necessari la utilització d'un intermediari entre l'aplicació del tomcat i el servidor de les captures. La utilització de dues màquines virtuals era important per no comprometre l'aplicació amb les consultes al servidor de captures. Per aquest motiu es va utilitzar dos servidors d'aplicacions: el Tomcat i

Jboss.

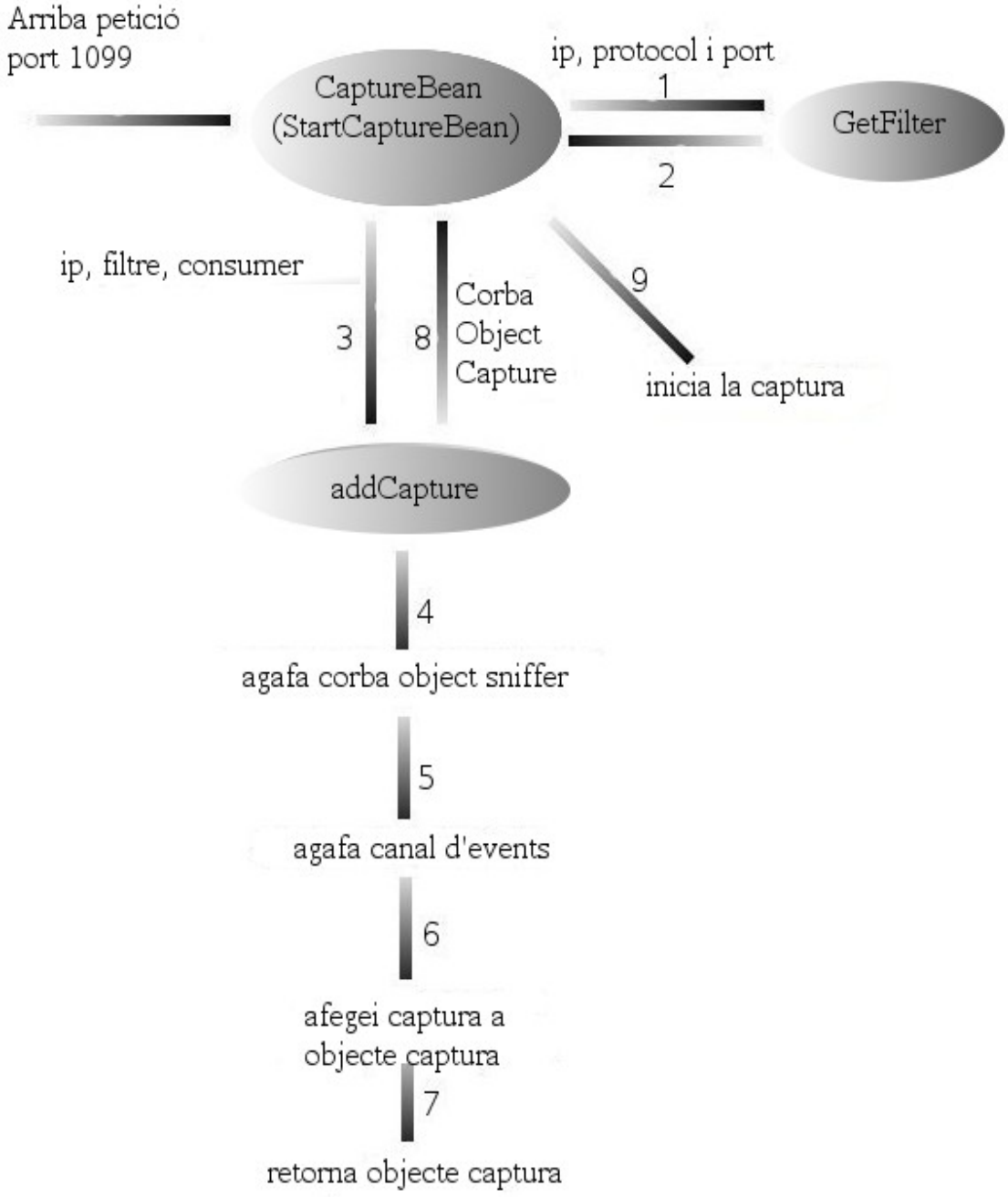


Figura 26: Petició d'una captura

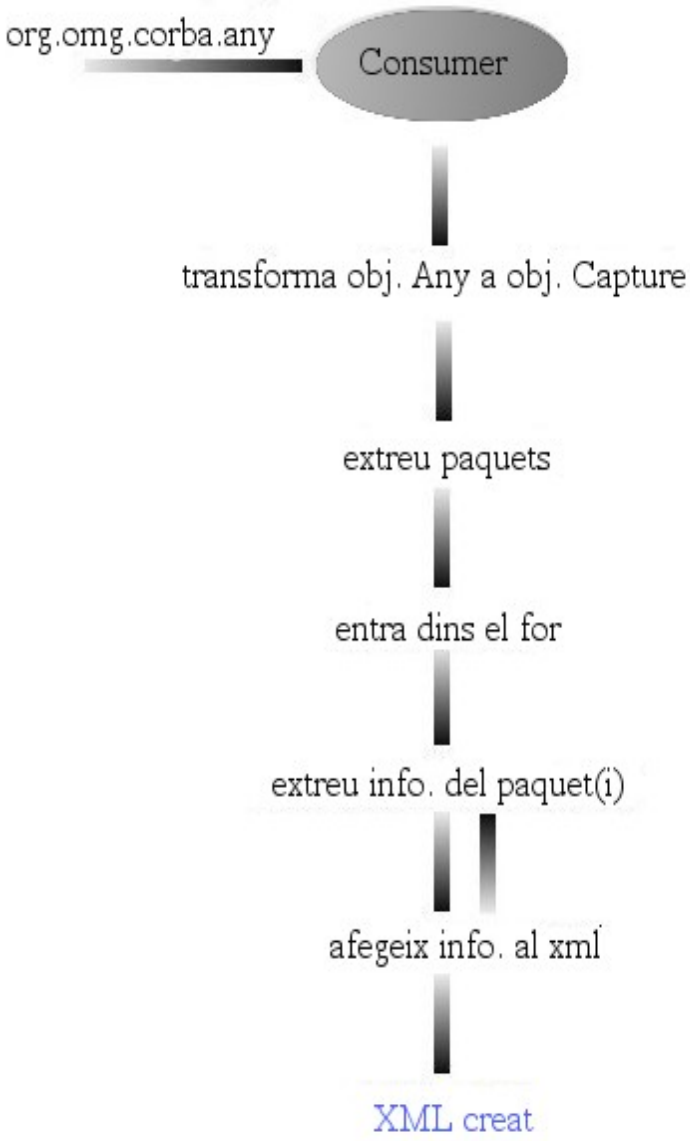


Figura 27: Passos d'una Captura

6 Implementació

6.1 Tecnologies utilitzades.

En el desenvolupament del projecte he utilitzat diferents tecnologies per aconseguir els objectius esperats. Abans d'escollir cada una de les tecnologies utilitzades, aquestes, van ésser meditades i van ésser escollides per un motiu. Tot seguit passaré a explicar perquè van ésser escollides unes i no unes altres.

6.1.1 Llenguatge de programació

La primera qüestió que ve al cap quan es vol desenvolupar una aplicació web és : en quin llenguatge la programaré?. Existeixen diferents llenguatges per a programar pàgines web : J2EE, php, .net, Sherpoint, etc. La meva elecció va ser **J2EE.**
Quan programes aplicacions web és molt important conèixer molt bé el llenguatge que utilitzes per programar-les; els seus avantatges i les seves limitacions. Necessitava utilitzar un llenguatge Opensource. Vaig haver d'escollir entre php i J2EE. En el meu cas coneixia bé tant php com J2EE. Em vaig decantar per J2EE ja que és un llenguatge més ordenat i més orientat a objectes que no pas php. En JAVA es poden definir classes privades, protected, etc. Cal dir, però que en les últimes versions de php es troben implementades aquestes funcionalitats. De tota manera no existeixen implementacions de CORBA per php; això conclou en la necessitat d'utilitzar J2EE. Un altre avantatge d'utilitzar J2EE és la possibilitat d'utilitzar frameworks els quals simplifiquen la feina i estructuren el codi de manera que sigui entendible per altres persones. En php també existeix algun framework com "PHPDAO" tot i que no donen tantes funcionalitats com els que existeixen per J2EE. Aquest últim avantatge es podria veure com un desavantatge si no es volgués un codi opensource, ja que qualsevol programador podria arribar a entendre el codi font.

6.1.2 Implementació de CORBA

Vaig haver d'escollir una **implementació de CORBA** per a comunicar el servidor de captures amb el client. Existeixen moltes implementacions de CORBA i cada una d'elles dóna uns serveis o uns altres. Alhora d'escollir quina d'elles em convenia més vaig haver d'estudiar quines eren les característiques del servidor de captures, quines les del client i quines eren les de la comunicació entre els dos.

Abans però explicaré, per sobre, els diferents que poden oferir les diferents implementacions:

El servei de noms ("**Naming Service**"): és la correspondència entre noms

convenients d'objectes i la referència a objectes reals

El servei cicle de vida ("**Object Life Cicle**"): s'encarrega de la creació, borrat, còpia i trasllat d'objectes corba.

El servei d' Esdeveniments ("**Events**"): s'encarrega de la notificació d'esdeveniments.

El servei d'Estat Persistent ("**Persistent State**"): s'encarrega de l'existència d'objectes a llarg plac i de la gestió del seu emmagatzemament.

El servei de Relació ("**RelationShip**"): s'encarrega de la gestió de representació i consistència de les relacions entres els objectes.

El servei d'Externalització ("**Externalization**"): Té la capacitat d'emmagatzemar la representació d'objectes a medis removibles i permeten més tard la seva reinternalització.

El servei de Transaccions ("**Transaction**"): s'encarrega de problemes de processament de transaccions comercials.

El servei de Temps ("**Time service**"): Ordena esdeveniments en el temps.

El servei de gestió ("**Management service**"): s'encarrega del control de versions.

El servei de Propietats ("**Property Service**"): Posa propietats a objectes corba en temps d'execució.

El servei de repositori ("**Interface Repository**"): S'encarrega de donar informació sobre objectes operació, com els paràmetres per invocar-l'ho.

El servei de Consulta ("**Query Service**"): Permet invocar als usuaris i als objectes consultes a objectes d'altres col·leccions.

Els serveis imprescindibles per a l'sniffer són el Servei de noms i el servei d'esdeveniments.

Respecte al servei de noms és obvi que la de tenir l'implementació escollida ja que sense ell no podríem cridar objectes del servidor. De fet, totes les implementacions de Corba disposen d'ell.

El servei d'esdeveniments també l'ha de tenir la implementació escollida. És molt important alhora de rebre les captures. El client ha de tenir un mètode que reculli i

processi les captures i s'ha d'activar quan es rebin captures per mitjà del canal d'esdeveniments. Aquest mètode que espera les captures per el canal d'esdeveniments rep el popular nom de consumidor.

Pel què fa als altres servies:

- El servei del **cicle de vida** no és molt important ja que en aquesta aplicació hi ha un sol esdeveniment que l'usuari des de l'aplicació llenca un sol cop. Per tant, no és necessari borrar esdeveniments del bus de peticions.

- El servei **Time Service** no és necessari ja que no serà necessari ni registrar esdeveniments periòdics ni controlar el temps d'execució.

- El servei d'**externalització** tampoc serà necessari ja que l'aplicació només necessita un ORB i per tant no serà necessari exportar objectes entre diferents orb's no connectats.

- El servei **Query Service** sempre és una ajuda extra ja que permet consultar els objectes corba i els seus serveis. Tot i això, no és un servei indispensable per a l'aplicació ja que, els objectes corba i els seus serveis, es poden esbrinar mirant el codi del servidor.

- El servei **Interface Repository** tot i que sempre és una ajuda poder consultar informació de les interfícies del servidor no és un servei prescindible pel mateix motiu que en el cas anterior.

- El servei **Property** no serà necessari ja que a l'aplicació no serà necessari posar propietats als objectes corba en temps d'execució. En el cas de realitzar una captura, l'objecte captura, no necessita propietats extra mentre s'executa ja que realitza una acció concreta que no necessita ésser modificada.

- El servei de **Persistència** en primera instància no és necessari ja que no cal guardar l'estat de l'objecte corba que realitza la captura. Tot i això, podria ser d'utilitat guardar dos estats de l'objecte captura: “en execució” i “stop” per saber del cert quan ha realitzat totes les captures. Per tant, tot hi no ésser indispensable seria bo que l'orb disposés d'ell.

- El servei de **Management** no s'utilitzarà ja que no s'implementarà cap tipus de control de versions.

- El servei de **Transacció** no serà necessari ja l'aplicació només utilitzarà un ORB.

- El servei de **Relació** no és necessari ja que no es definiran relacions entre els objectes Corba de l'aplicació. En un futur, si es vol ampliar l'aplicació amb més funcionalitats podria ésser d'utilitat.

A la següent taula (Figura 28) podem veure amb les característiques de les diferents implementacions de Corba més conegudes:

Figura 28: Característiques de diferents implementacions de CORBA

Producte		*Language Mappings*										*Protocols*		*Core*			
	OS	*CPP*	*C*	*St*	*Ada*	*Java*	*Lisp*	*Cobol*	*Python*	*Tcl*	*Perl*	*SOAP*	*COM*	*Dll*	*DSI*	*IFR*	*BOA*
BOABorland	-	Y	-	-	-	Y	-	-	-	-	-	Y	-	Y	Y	Y	Y
BusinessWare	-	Y	-	-	-	Y	-	-	-	-	-	P	Y	Y	Y	Y	P
DSTC Corba	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
e*ORB C++	-		Y	Y	?	?	?	?	?	?	?	?	?	?	-	-	-
Egypt/Taiwan	-	P	Y	-	-	-	-	-	-	-	P	-	-	Y	Y	Y	Y
EJCCM	Y	-	-	-	-	Y	-	-	-	-	-	-	-	-	-	-	-
Eorb	-	Y	Y	-	-	Y	-	-	-	-	-	-	-	Y	Y	-	-
Gibraltars	Y	P	Y	-	-	-	-	-	-	-	P	-	-	-	-	-	-
GNU Classpath	Y	-	-	-	-	Y	-	-	-	-	P	-	-	Y	Y	-	-
HP Nonstop CORBA	-	Y	-	-	-	Y	-					Y	-	Y	Y	Y	-
JacORB	Y	-	-	-	-	Y	-	-	-	-	-	-	-	Y	Y	Y	-
Jonathan	Y	-	-	-	-	Y	-							Y	Y	P	-
MICO	Y	Y	-	-	-	-	-	-	-	Y	-	-	-	Y	Y	Y	Y
microORB	P	-	Y	-	P	-	-	-	-	-	-	-	-	-	-	-	-
MiddCor	-	-	-	-	-	-	-	-	-	-	-	?	?	-	-	-	-
OmniORB 3	Y	Y	-	-	-	-	-	-	Y	-	-	?	?	Y	Y	Y	Y
Openfusion TCS	Y	Y	P	-	-	Y	-	-	-	-	-	Y	-	Y	Y	Y	Y
OpenORB	Y	-	-	-	-	Y	-	-	-	-	-	Y	-	Y	Y	Y	Y
ORBacus 4	-	Y	-	-	-	Y	-	-						Y	Y	Y	-
Orbexpress GT	-	Y	-	-	-	-	-	-	-	-	-	-	-	P	P	P	-
Orbix	-	Y	-	-	-	Y	-	P	-	-	-	Y	Y	Y	Y	Y	-
Orbriver RT	-	Y	-	-	Y	Y	-	-	-	-	-	-	-	Y	P	Y	-
PolyORB	Y	-	-	-	Y	-	-	-	-	-	-	Y	-	Y	Y	Y	-
Sankhya	-	Y	-	-	-	Y	-	-	-	-	-	Y	-	Y	Y	Y	-
SmalltalkBroker	-	-	-	Y	-	-	-	-	-	-	-	-	-	Y	Y	Y	-
The ACE ORB	Y	Y	-	-	-	-	-	-	-	-	-	-	-	Y	P	Y	-
Tuxedo 8.0	-	Y	-	-	-	Y	-	-	-	-	-	-	Y	Y	Y	Y	-
VBOrb	Y	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Y	-
Visibroker RT	-	P	-	-	-	-	-	P	-	?	Y	-	-	Y	Y	Y	Y

Abbrev.	Descripció
Y	Característica present
N	Característica no present
P	Característica de futur
?	No se sap si la característica hi és present
OS	Open Source ORB
St	“Samalttalk”
IIOP	Protocol Internet Inter-ORB
SOAP	Protocol d'Accés a un Objecte Simple
COM	Model d'Objecte Comú
DII	Interfície Dinámica d'Invocació
DSI	Interfície d'Esquelet Dinàmica
IFR	Interfície de Repositori
BOA	Adaptador d'Objecte Bàsic

	Core								Services										
Producte	*POA*	*VTS*	*Min*	*AMI*	*CCM*	*QoS*	*FT*	*RT*	*PSS*	*CONC*	*PROP*	*EVNT*	*RLSH*	*NAM*	*SEC*	*TIME*	*TRAD*	*NOTF*	*TRANS*
BOABorland	Y	Y	-	-	-	Y	Y	-	-	-	-	Y	-	Y	Y	-	-	Y	Y
BusinessWare	P	P	?	Y	?	?	?	?	?	?	?	?	?	Y	?	?	?	?	?
DSTC Corba	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Y	-
e*ORB C++	Y	-	Y	Y	Y	?	Y	Y	Y	-	-	Y	-	Y	Y	Y	-	Y	?
Egypt/Taiwan	Y	-	-	-	-	Y	Y	Y	Y	Y	-	Y	Y	Y	Y	Y	Y	Y	Y
EJCCM	-	-	-	-	Y	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Eorb	Y	-	Y	-	-	-	Y	Y	-	-	-	-	-	Y	-	-	-	Y	-
Gibraltars	Y	Y	Y	-	-	-	-	-	-	-	Y	Y	-	Y	Y	Y	Y	Y	-
GNU Classpath	Y	Y	-	-	-	P	P	P	-	-	-	-	-	Y	-	-	-	-	-
HP Nonstop CORBA	Y	Y	-	P	-	Y	Y	-	-	-	P	Y	-	Y	P	P	P	P	Y
JacORB	Y	Y	-	Y	-	-	-	-	-	Y	-	Y	-	Y	P	-	Y	Y	Y
Jonathan	P	P	-	Y	P	-	-	-	P	-	-	Y	-	Y	-	-	-	-	P
MICO	Y	Y	Y	P	Y	-	P	-	-	-	Y	Y	Y	Y	Y	Y	Y	P	-
microORB	Y	-	Y	Y	-	Y	-	Y	-	-	-	-	-	Y	-	-	Y	P	-
MiddCor	Y	Y	-	-	-	-	-	-	?	?	?	Y	?	Y	?	?	?	Y	?
OmniORB 3	Y	-	-	-	-	-	-	-	-	-	-	Y	-	Y	-	-	-	Y	-
Openfusion TCS	Y	Y	Y	Y	P	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P
OpenORB	Y	Y	-	-	-	-	P	-	Y	Y	Y	Y	-	Y	-	Y	Y	Y	Y
ORBacus 4	Y	Y	-	P	-	-	-	-	-	-	Y	Y	-	Y	-	Y	Y	Y	Y
Orbexpress GT	Y	P	Y	Y	P	Y	-	-	-	-	-	Y	-	Y	P	P	-	P	-
Orbix	Y	Y	-	Y	-	Y	Y	-	Y	-	-	Y	-	Y	-	-	Y	Y	Y
Orbriver RT	Y	Y	P	P	P	P	P	P	-	-	-	Y	-	Y	P	P	-	P	-
PolyORB	Y	P	-	-	-	P	P	Y	-	-	-	Y	-	Y	P	Y	-	Y	-
Sankhya	Y	P	Y	P	-	P	Y	P	P	-	-	Y	-	Y	Y	-	P	P	-
SmalltalkBroker	-	-	-	Y	-	-	-	-	-	Y	-	Y	-	Y	-	-	-	-	Y
The ACE ORB	Y	Y	Y	Y	P	Y	P	P	-	Y	Y	Y	-	Y	P	Y	Y	Y	-
Tuxedo 8.0	Y	Y	-	Y	-	Y	Y	-	-	Y	-	Y	-	Y	Y	-	-	Y	Y
VBOrb	-	Y	-	-	-	-	-	-	-	-	-	-	-	Y	-	-	-	-	-
Visibroker RT	Y	Y	Y	Y	-	-	-	Y	-	-	?	Y	-	Y	-	-	P	-	-

Abbrev.	Descripció
AMI	Interfície de Missatges Asíncrona
CCM	CORBA Component Model
QoS	Quality of Service
FT	Tolerància a Errades
RT	Temps Real
PSS	Servei d'Estat Persistent
CONC	Servei de Concurrència
PROP	Servei de Propietats
EVNT	Servei d'Esdeveniments
RLSH	Servei de Relacions
NAM	Servei de Noms
SEC	Servei de Seguretat
TIME	Servei de Temps
TRAD	Servei de Comerç
NOTF	Servei de Notificacions
TRANS	Servei de Transaccions
POA	Apadtador d'Objecte Portable
VTS	Valor de Tipu Semàntic
Min	MinimumCORBA

6.1.3 Frameworks

Realitzar una aplicació amb java sense cap framework és molt costós. Per aquest motiu vaig decidir utilitzar **struts**, un framework sota la plataforma JAVA i sota el patró MVC.

En el mercat trobem molts frameworks spring, JSPWidget, struts, SwingWeb i de fet hi ha moltes aplicacions que utilitzen més d'un framework per tal d'aprofitar alguna cosa de cada un d'ells. Struts ja era conegut per mi i tenia la certesa que és realment eficaç alhora d'estalviar temps de desenvolupament en un projecte.

A l'aplicació no s'utilitza base de dades. Tot i això en un futur es podria necessitar. Per tant vaig incorporar al projecte **Hibernate,** un altre framework que facilita treballar amb bases de dades.

6.1.4 Servidors

L'aplicació utilitza dos servidors **Tomcat** i **JBOSS.** El motiu d'utilitzar dos servidors és senzill l'aplicació conté un “ejb” el qual ha de corre en una màquina virtual diferent de la màquina virtual de l'aplicació. A més a més el tomcat no pot corre ejb's i JBOSS si.

6.1.5 XML

Les captures processades es guarden en un xml per tal de l'aplicació, des de Tomcat, les recuperi. Per manipular i crear els xml's s'ha utilitzar una llibreria molt eficaç **JDOM**. Vaig escollir aquesta ja que permet de manera senzilla crear XML's i a més a més hi ha molta documentació a la xarxa.

6.1.6 Ajax

Vaig escollir utilitzar **ajax** sense utilitzar-ne cap framework ja que la tasca que calia fer no era complicada i per tant era implementable sense la necessitat de cap framework. La tasca és crea un xml amb informació de les captures i més tard agafar aquesta informació del fitxer xml.

6.1.7 Displaytag

És una llibreria necessària per mostrar taules de dades dinàmiques. Les taules amb les dades de les captures han d'ésser dinàmiques per poder anar incorporant les captures que arriben del servidor. Aquesta llibreria simplifica la feina del programador ja que disposa d'eines molt potents per aconseguir aquest objectiu i el

programador no ha d'implementar el codi des de zero.

6.1.8 Entorn integrat

Dins de les diferents opcions que trobem en el mercat vaig escollir l'**eclipse** ja que he treballat amb aquest entorn integrat i el conec bé.

6.2 Detalls d'implementació

6.2.1 Instal·lar el servidor de captures

Cal dir que aquesta aplicació ha estat instal·lada sobre Linux Ubuntu i Linux Suse 11.0. En cada cas, pot variar la manera d'instal·lar les llibreries i packages. Per exemple, amb el linux Suse les llibreries del repositori del sistema operatiu s'instal·len accedint al yast(Software Management) com a administrador i en ubuntu seria "sudo apt-get install (nom del package)".
-Instal·lar aquest servidor no va ser fàcil. Primerament vaig haver d'instal·lar les llibreries MICO, OpneSSL i PcapLib.

A continuació trobem la descripció dels passos què vaig seguir:

Primerament caldrà mirar si tenim instal·lat el package "make", necessari per instal·lar totes les llibreries. En cas negatiu sol trobar-se en el repositori de SO.

Podem mirar si tenim instal·lades les llibreries MICO,OpenSSL i PCapLib executant: "openssl version" , "tcpdump -help" i "idl -version" , respectivament. En cas afirmatiu no serà necessari instal·lar-les.

-Ara ja ens podem posar a instal·lar OpenSSl. Les comandes que vaig seguir són:

- *./config – prefix=/usr/local/openssl shared zlib-dynamic*
- *make*
- *make test*
- *make install*
- *ldconfig*

(Moltes versions no eren compatibles amb Suse 11.0, al final vaig instal·lar la versio 0.9.8g canviant -m486 per -mtune=i486 de l'arxiu "Makefile". Vaig haver d'instalar la llibreria zlib-devel del repositori del SO i afegir-la a la configuració).

-Per instal·lar MICO és necessari tenir instal·lat:

- gnu make version 3.7 or newer (required)
- C++ compiler and library (required):
  - g++ 2.7.2.x and libg++ 2.7.2, or
  - g++ 2.8.x and libg++ 2.8.x, or
  - egcs 1.x
- flex 2.5.2 or newer (optional)
- bison 1.22 or newer (optional)
- JDK 1.1.5 (SUN's Java developers kit) (optional)
- JavaCUP 0.10g (parser generator for Java) (optional)

Vaig executar el següent:

- *./configure –with-qt=/usr/local/qt*
- *gmake*
- *gmake install*
- *ldconfig*

(Vaig tenir problemes per instal·lar mico degut a la versió finalment vaig vaig instal·lar la versió 2.3.13)

-Per instal·lar Pcaplib, primer cal instal·lar, utilitzant el repositori de packages del SO, “bison” i “flex”. Es pot, però, configurar pcaplib sense “flex” i sense “bison” amb aquesta comanda “./configure --without-flex –without-bison”. Després del configure es crea el make i executem:

- *make*
- *make install*

(Cal haver instal·lat tcpdump i tenir-l'ho a la mateixa carpeta que libpcap. La versió que vaig instal·lar és la 0.9.8)
-Instal·lar tcpdump (versió 3.9.8), en cas de no tenir-l'ho, és necessari instal·lar del repositori del SO la llibreria devel-pcab. Executar:

- *./configure*
- *make*
- *make install*

- Per a la integració amb l'aplicació web és necessari tenir instal·lat el jdk-1.6 o superior . Dins del jre (Runtime time environment) s'ha d'instal·lar OpenORB

(la implementació de CORBA versió 1.4.0 o superior i) i el package Tools d'OpenORB versió 1.4.0 . Cal tenir en compte que la versió de l'OpenORB i de Tools ha de ser la mateixa. Un cop installat s'han d'agafar els jars del lib de OpenORB i Tools a la carpeta "/lib/ext/" del jre. Cal , també instal·lar els serveis NameService i EventService de l'OpenORB dins del jre. Si, en el futur es necessitessin més serveis es poden instal·lar però els ja esmentats són imprescindibles.

Dins de l'OpenORB troba un fitxer anomenat "orb.properties". Aquest s'ha de posar dins del lib del jre amb les línies:

"org.omg.CORBA.ORBClass=org.openorb.CORBA.ORB" i
"org.omg.CORBA.ORBSingletonClass=org.openorb.CORBA.ORBSingleton".

Ara ja tenim tot l'entorn preparat per instal·lar el codi del servidor de captures. El primer que hem de fer és baixar el codi del servidor del csv amb les comandes "cvs -d pserver:anonymous@cvs.sf.net:/cvsroot/sniffit login" i
"cvs -d pserver:anonymous@cvs.sf.net:/cvsroot/sniffit checkout server-alpha" si s'utilitza el csv de linux. Es pot utilitzar el "tortoise" ò es pot utilitzar l'eclipse, el qual té una vista que s'anomena CSV Exploring cal crear una nou repositori location amb les següents dades:

- *:pserver:anonymous@anonymous@cvs.sf.net:/cvsroot/sniffit*
- *Connection Type -> pserver*
- *User ->anonymous*
- *Password ->anonymous*
- *Host ->anonymous@cvs.sf.net*
- *Repository path ->/cvsroot/sniffit*

Un cop descargat el codi cal, des de la carpeta server-beta executar les següents comandes:

- *aclocal*
- *autoconf*
- *automake -a*
- *./configure*
- *make*

Un cop executat el dimoni sniffer es troba a server-beta/daemons/sniffer. No cal compilar els fitxers idl del servidor ja que l'apicació web ja conté les classes necessàries per comunicar-se amb el dimoni sniffer. De totes maneres si es volguessin realitzar canvis en aquest servidor si seria necessari compilar aquests arxius. Per si calgués compilar-los les comandes necessàries són:

dins la carpeta header:
*java org.openorb.compiler.IdlCompiler -d <path del cliente web> *.idl*

dins la carpeta types:
*java.org.openorb.compiler.IdlCompiler -d <path del cliente> *.idl*

dins la carpeta common:
*java org.openorb.compiler.IdlCompiler -d <path del cliente> -I ../types *.idl*

dins la carpeta sniffer
*java.org.openorb.compiler.IdlCompiler -d <path del cliente> -I../types -I ../common -I ../../header *.idl*

Dins el codi del servidor trobem altres dimonis que la l'aplicació no utilitza. Per tant no seria necessari compilar-los. El path del client hauria de ser la carpeta src de l'ejb anomenat Captures.

Finalment, per posar en funcionament el servidor primer cal llençar el servei de noms amb la comanda:

nsd -ORBIIOPAddr inet:<maquina>:<port>

i el servei d-events amb la comanda:

eventd -ORBInitRefNameService=corbaloc::<maquina>:<port>/NameService

El servidor es llenca amb la comanda:

snifferd -ORBInitRef NameService=corbaloc::<maquina>:<puerto>/NameService

Podem afegir *-ORBDebug All* per veure les traces de log. Existeix diferents nivells de log per jo vaig utilitzar “All” per veure tots els nivells.

Cal comentar també que és necessari llençar el servidor com a superusuari per no tenir problemes de permisos. Si no ho fem podria no realitzar captures.

6.2.2 Instal·lar servidor Jboss i Tomcat

No és complicat, tant sols cal descomprimir els packages de Tomcat i de Jboss cada un dins de la mateixa carpeta de la màquina que albergarà l'aplicació. Cal posar el

war de l'aplicació web dins de la carpeta webapps del tomcat i el jar de l'ejb dins de la carpeta “server/default/deploy” de Jboss.

6.2.2.1 Integració dels servidors

Tomcat i JBOSS no estan configurats per treballar conjuntament. Hi ha una part de la configuració d'un ò de l'altre que s'ha de modificar. En el meu cas vaig optar per realitzar els canvis a JBOSS ja que el servidor Tomcat és el que alberga l'aplicació web en si i el port, per defecte de tomcat, per tràfic http és el 8080 el qual s'acostuma a utilitzar. Per tant cal realitzar uns canvis a la configuració de JBOSS perquè no hi hagi col·lisió entre els ports que utilitza cada un.

Cal accedir a “$JBOSS_HOME/server/default/deploy/jbossweb/ tomcatxx.sar/server.xml” i el connector HTTP per defecte veiem que és 8080. Cal canviar-l'ho per exemple al 8090. En el mateix arxiu cal canviar també el connector AJP que normalment ve per defecte al port 8009 al port 8019, per exemple. D'aquesta manera no hi haurà conflictes amb els ports del tomcat, que per defecte, els quals solen ésser els mateixos.

Cal accedir també a l'arxiu “$JBOSS_HOME/server/default/deploy/http-invoker.sar/ META-INF/jboss-service.xml” en el qual cal cal canviar el port de l'atribut invoKerURLSuffix i posar el seu valor a 8090, per exemple. De fet, haurà d'ésser el mateix port que hem posat al connector HTTP de l'anterior arxiu.

Per altre banda tomcat i JBOSS es comunicaran per el port 1099 (jnp:localhost:1099”) el qual es pot modificar dins de JBOSS a “JBOSS_HOME/server/deploy/naming.sar/META-INF/jboss-service.xml”. Cal tenir en compte que si es canvia aquest port cal canviar-l'ho en codi de l'aplicació web que estableix comunicació amb JBOSS.

6.2.2.2 Lookup de Tomcat a Jboss

Per aconseguir establir comunicació entre tomcat i jboss vaig tenir algun problema. Principalment per inicialitzar el contexte per realitzar posteriorment el lookup. Per aconseguir-ho va ser necessari moure tots els jars de “/jboss/client” al lib de l'aplicació “/WEB-INF/lib/”. Això va produir algun conflicte amb jars que dins de Tomcat no funcionaven correctament . Vaig haver de buscar altres versions dels jars amb conflictes.

També vaig tenir problemes amb alguna llibreria del projecte. Algun jar no era compatible amb jboss.5.0.1.GA i llavors l'establiment de la connexió es trencava ja

que jboss no trobava les classes necessàries per realitzar-la. Per exemple la llibreria "javaee.jar" la vaig canviar per "geronimo-spec.2.1".

Vaig tenir problemes de "CastException" al fer el lookup. Això va ser degut a que tenia el jar de l'ejb a "/WEB-INF/lib/" i no en el classpath. La solució va ser incorporar-l'ho al classpath i no tenir-l'ho duplicat a més d'un lloc.

Un cop arreglat aquest problema va ser possible establir comunicació.

6.2.3 Comunicació Jboss amb el servidor de captures

Per aconseguir establir la connexió entre aquests dos element és necessari que els jars d'OpenORB es trobin en el jre. Per altre banda quan es compilen els idl del servidor cal posar com a ruta del client, la ruta del ruta de l'ejb que correrà sota JBOSS. Aquests arxius compilats també es troben a l'aplicació que corre sota TOMCAT per si fossin necessaris.

Abans de cridar qualsevol servei del servidor de captures cal inicialitzar l'ORB, el Servei de noms, el servei d'esdeveniments i els dimonis, cal registrar les factories necessàries per a la comunicació i finalment llençar l'ORB. En aquest moment encara no s'ha establert comunicació però l'entorn ja està preparat per a què sigui possible. Dins de l'ejb es realitzen dues crides al servidor en moments diferents: per esbrinar els sniffers de la xarxa i per realitzar les captures. L'entorn de comunicació s'estableix durant la primera crida al servidor i després ja es manté fins que es pari el servidor de l'ejb.

6.2.4 Implementació de l'EJB

Per implementar l'ejb vaig escollir la última versió: la versió 3.0 . Aquesta última

versió té avantatges sobre les anteriors versions. Aquestes són per exemple que no és necessari utilitzar fitxers xml de configuració, fa més fàcil la implementació de la persistència mitjançant la api JPA. Jo vaig crear un ejb Stateless(Sense estat); ideals per implementar la lògica de negoci sense estat, la qual es pot invocar simultàniament

per múltiples clients. Dins aquest ejb tenim dues interfícies: una local i una altre remota. Les classes i els mètodes que vaig implementar per realitzar la captura es troben a l'interfície remota en la qual es defineixen els objectes que s'invoquen des de fora del servidor. Cal tenir en compte, però, que alhora d'implementar una interfície remota cal que tots els seus objectes implementin la classe "serializable".

Cal dir que eclipse simplifica feina alhora de crear l'ejb ja que amb el plugin crea tot

sol les classes necessàries de l'ejb i només cal implementar-les.

La tasca principal de l'ejb és cridar al servidor de captures i guardar-les en un xml. Vaig tenir alguns problemes amb la llibreria JDOM necessària per crear l'xml. Vaig haver d'afegir-la al lib de JBOSS, "server/default/deploy/lib/". La corba d'aprenentatge va ser ràpida. És un sofware senzill i ràpid d'utilitzar.

7 Proves de sistema

7.1 Disseny

Un cop acabada l'aplicació web cal comprovar que totes les funcionalitats desenvolupades funcionen i compleixen els requisits establerts per l'usuari. Per aquest motiu he pensat en una sèrie de proves que ha de passar l'aplicació.

1.- La primera prova és entrar a l'aplicació i realitzar una primera configuració de la captura a realitzar i realitzar unes quantes captures. Les dades a configurar poden ésser ip:(ip de la màquina), port: 80 (per aquest port la màquina estableix comunicació amb internet), protocol: tcp, número de paquests: 1. Les altres dades no són rellevants per realitzar la comprovació.

2.- La segona prova tracta de realitzar una captura amb el checkbox de “Localhost”. El numero de paquets capturats el posarem a 5 i realitzarem les captures a qualsevol port.

3.- Per realitzar captures a diferents ports tan sols cal separar-l'hos amb una coma. En aquesta prova provarem si funciona posant captures a diferents ports. Introduirem: “21,22,23,80,8080,9090”.

4.- Ara ja em provat que l'aplicació realitza captures complint els requisits d'usuari. Aquest prova consisteix en realitzar una captura simple, guardar-la i tornar-la a obrir.

5.- La cinquena prova consisteix en realitzar una reconfiguració de les captures a realitzar i veure que realment la nova captura compleix els requisits esperats. Per realitzar aquesta prova primer em realitzat un captura de 4 paquets per el port 80 amb el checkbox de tots els protocols activat i després realitzem una reconfiguració amb 6 paquests per el port 8080 només amb tráfic ip.

6.- La sisena prova planejada és realitzar una captura amb les mateixes dades què a la prova 1 exceptuant el un número de captures el qual escollirem un número gran per esbrinar si algun dels servidors queda saturat. Comencarem per 50 captures i anirem pujant el número de captures fins a col·lapsar algun servidor.

7.- L'aplicació esbrina quants sniffers hi ha a la xarxa. Aquesta prova doncs consisteix

en instal·lar l'sniffer en dues màquines diferents les quals estiguin en xarxa i veure

que realment podem accedir a l'altre sniffer i realitzar captures. Per realitzar aquesta prova caldrà disposar de dos ordinadors en xarxa i que disposin de linux. Un cop instal·lat l'sniffer caldrà deshabilitar el firewall dels dos ordinadors.

8.- En aquesta prova obrir tres pestanyes al navegador i realitzarem tres captures alhora. D'aquesta manera esbrinarem si hi ha algun problema amb els servidors.

7.2 Resultats

1.- El resultat de la primera prova ha estat positiu. L'sniffer ha filtrat correctament les captures.

Les dades entrades són les què podem veure a la següent imatge (Figura 29):

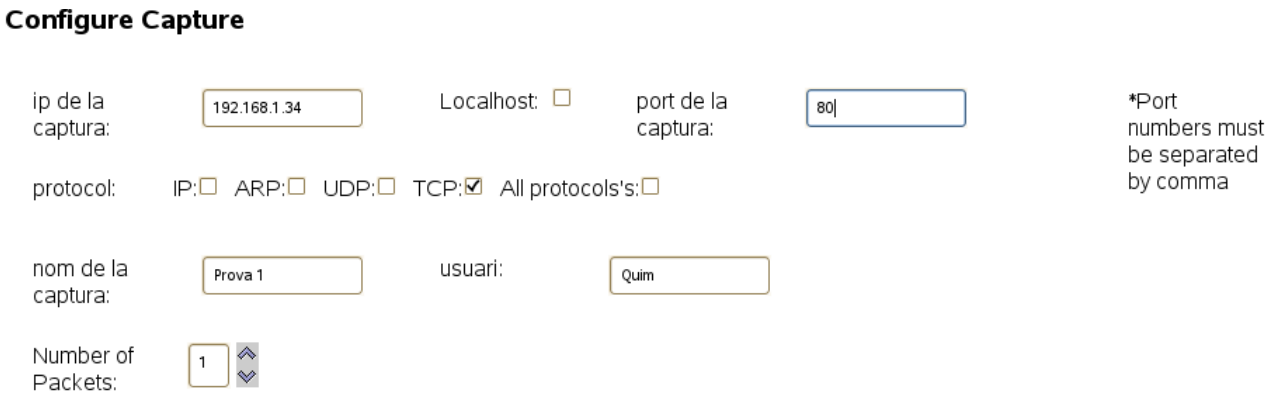


Figura 29: Configuració

Els resultats de la prova els podem veure a la següent imatge (Figura 30):

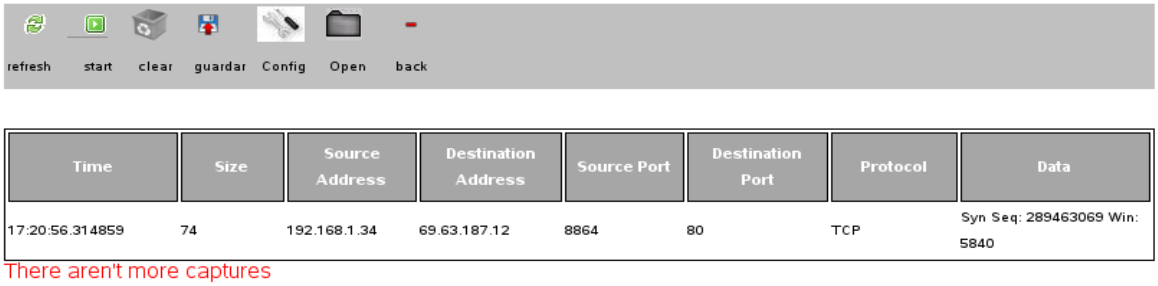


Figura 30: Captura resultant

2.- El resultat de la segona prova a estat positiu ja que l'aplicació ens ha mostrat els resultats esperats. De fet, el resultat ha estat el mateix que el de la prova 1, tal i com s'esperava.

Les dades configurades a l'aplicació són les de la següent imatge (Figura 31):

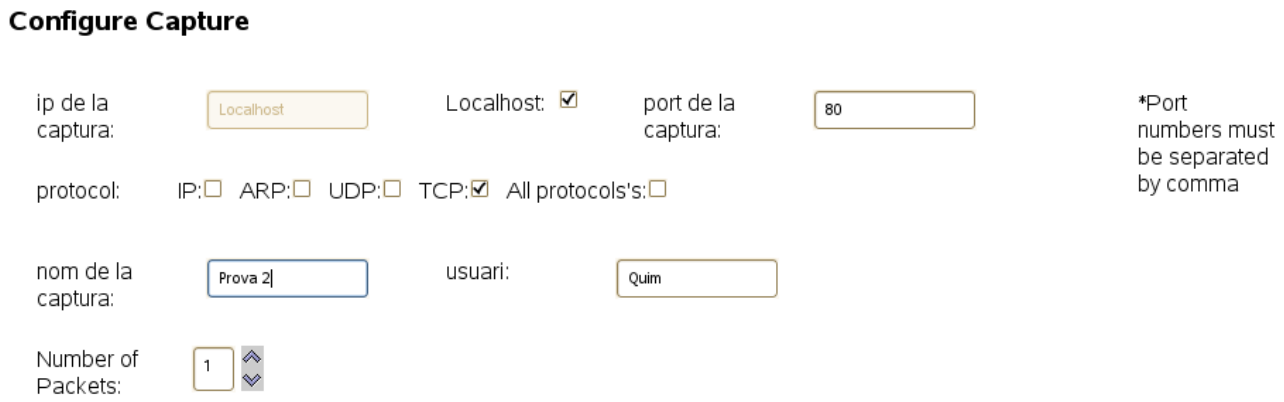


Figura 31: Configuració

3.- El resultat de la tercera prova ha estat positiu. Tot i això el resultat que surt per pantalla pot portar a pensar que no ha estat així ja que no per tots els ports trobem tràfic. Per tant, per comprovar que realment l'sniffer funciona cal fer un print per la consola de la comanda que es passa a la llibreria pcap. Un cop feta aquesta comprovació veiem que és:

ip or arp or udp or tcp and host 192.168.1.34 and port 21 or port 22 or port 23 or port 80 or port 8080

Per tant verifiquem que el funcionament és correcte.

Les dades introduïdes per realitzar la prova són les següents (Figura 32):

SNIFFER

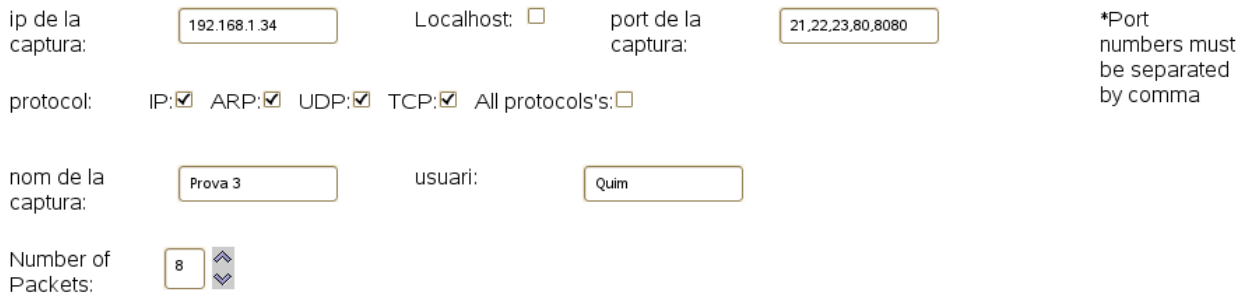


Figura 32: Configuració

El resultat obtingut és el següent (Figura 33):

Time	Size	Source Address	Destination Address	Source Port	Destination Port	Protocol	Data
16:18:30.3352	1506	74.125.8.28	192.168.1.34	80	4086	TCP	Ack 654971006 Seq: 1996562735 Win: 181
16:18:30.196	66	192.168.1.34	74.125.8.28	4086	80	TCP	Ack 1996562735 Seq: 654971006 Win: 1530
16:18:30.154	1506	74.125.8.28	192.168.1.34	80	4086	TCP	Ack 654971006 Seq: 1996561295 Win: 181
16:18:29.996767	66	192.168.1.34	74.125.8.28	4086	80	TCP	Ack 1996561295 Seq: 654971006 Win: 1530
16:18:29.996721	1506	74.125.8.28	192.168.1.34	80	4086	TCP	Ack 654971006 Seq: 1996559855 Win: 181
16:17:4.501573	74	192.168.1.34	209.85.227.101	22095	80	TCP	Syn Seq: 650485275 Win: 5840
16:3:51.99080	74	192.168.1.34	74.125.10.164	24782	80	TCP	Syn Seq: 1085077800 Win: 5840
16:1:35.248524	74	192.168.1.34	69.63.187.17	26505	80	TCP	Syn Seq: 1100805919 Win: 5840

There aren't more captures

Figura 33: Llistat de captures

Com podem observar en el resultat tot i haver introduït molts ports els primers 8 paquets han estat com era d'esperar per el port 80, el de tràfic web.

4.- El resultat d'aquesta prova ha estat positiu.

A la següent imatge es pot veure com es guarden les captures amb el nom de prova_4 (Figura 34):

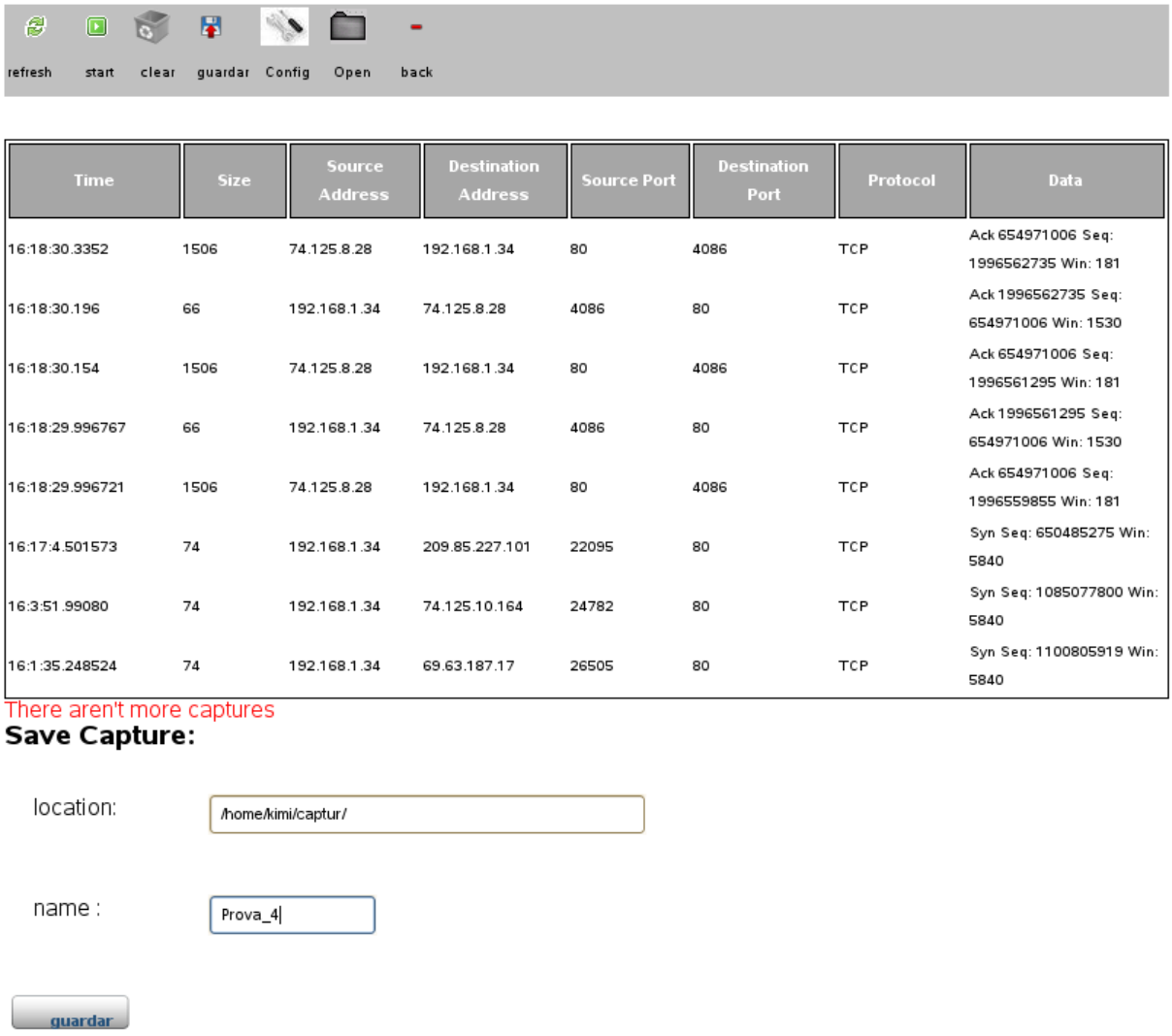


Figura 34: Llistat de captures obertes

A la següent imatge podem veure com obrim l'xml guardat prèviament (Figura 35):

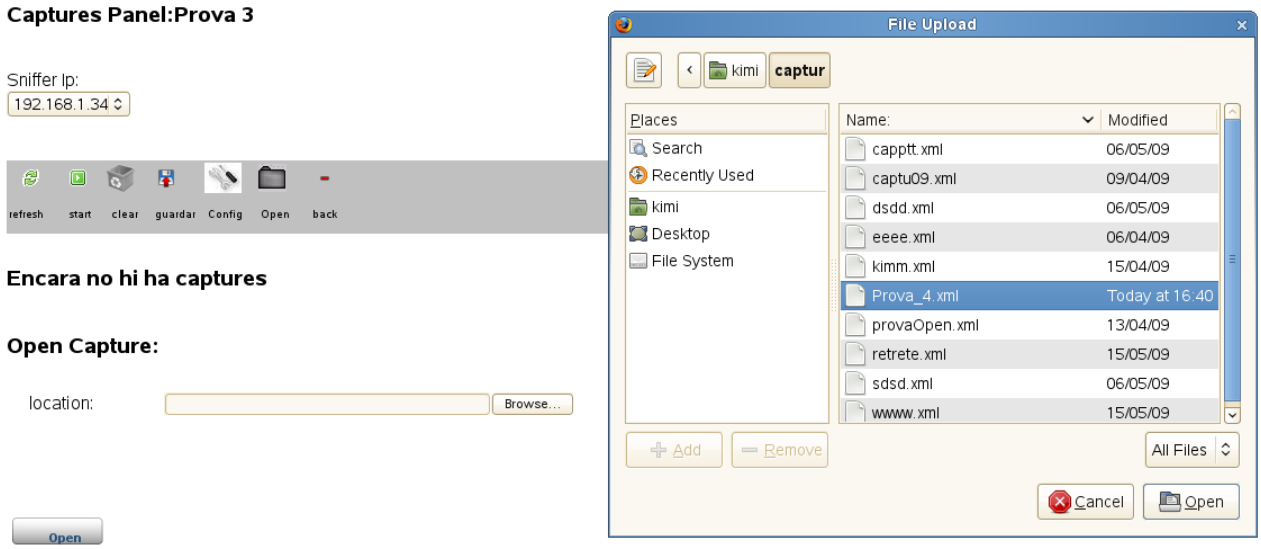


Figura 35: Obrir les captures

5.- El resultat d'aquesta prova ha estat positiu. Per comprovar el bon funcionament he utilitzat la comprovació visual dels resultats i la comprovació del filtre creat que es passa a la llibreria pcap.

En la primera configuració el filtre és:

ip or arp or udp or tcp and host 192.168.1.34 and port 80

En la segona configuració el filtre és:

ip and host 192.168.1.34 and port 80 80

6.- El resultat ha estat positiu. Tot i això l'aplicació molts cops detecta un desbordament del tamany d'un paquet i atura la captura. Si no es detecta un desbordament es poden arribar a fer unes 250 captures.

7.- El resultat d'aquesta prova ha estat positiu. Es va instal·lar l'aplicació a dos ordinadors en xarxa i es va comprovar que es podia accedir a l'altre sniffer i realitzar captures remotament.

8.- El servidor, com era d'esperar, tant sols agafa una de les tres peticions ja que no

estan implementades cues d'espera. Per altre banda, els resultats es mostren a les tres pestanyes del navegador ja que totes accedeixen al mateix xml i al mateix element de la sessió.

8 Anàlisi econòmic

Com ja s'ha comentat anteriorment a l'apartat 4 l'anàlisi econòmic es divideix en tres parts: el cost del hardware, el cost del sofware, i els cost humà. Amb el projecte ja acabat puc dir que de les tres variables la única què a patit canvis és la part que referència els costs dels recursos humans. Abans de començar el projecte es va fer una previsió temporal de les diferents etapes d'aquest però aquestes han sofert enderreriments en la majoria de les etapes.

8.1 Anàlisi temporal

Tal com he comentat en la introducció d'aquest apartat i va haver enderreriments respecte el pla temporal calculat inicialment. Tot seguit mostro l'evolució temporal real del projecte (Figura 36).

Nom Tasca	Data d'inici	Data de final
T1.- Baixar sniffer i provar-l'ho	25/09/08	29/09/08
T2.- Fer funcionar l'sniffer i entendre'l	29/09/08	27/12/08
T2.1.- Obrir l'aplicació	29/09/08	25/10/08
T2.2.- Realitzar captures	26/10/08	27/12/08
T3.- Montar entorn de la nova aplicació	28/12/08	11/01/09
T4.- Comunicació entre client (Tomcat) i el servidor de captures sense utilitzar ejb	12/01/09	25/01/09
T5.- Recuperar captures sense ejb	26/01/09	30/01/09
T6.- Montar ejb 3.0 per realitzar captures	31/01/09	25/02/09
T6.1.- Montar jboss en una màquina virtual diferent de la de Tomcat	31/01/09	02/02/09
T6.2.- Establir connexió entre Tomcat i jboss	03/02/09	13/02/09
T6.3.- Establir connexió entre jboss i servidor de captures	14/02/09	25/02/09
T7.- Recuperar captures	26/02/09	09/03/09
T7.1.- Agafar captures provinents del servidor de captures	26/02/09	28/02/09
T7.2.- Utilitzar JDOM per crear XML amb captures.	01/03/09	09/03/09
T8.- Mostrar les captures	09/03/09	27/03/09
T8.1.- Accedir al XML mitjançant ajax i montar la request amb les captures	09/03/09	17/03/09
T8.2.- Utilitzar displaytag per mostrar les captures de forma dinàmica	18/03/09	27/03/09
T9.- Implementar funcionalitats extra	28/03/09	16/05/09
T9.1.- Guardar captures	28/03/09	08/04/09
T9.2.- Borrar captures	09/04/09	13/04/09
T9.3.- Reconfigurar captures	14/04/09	21/04/09
T9.4.- Obrir captures guardades	22/04/09	02/05/09
T9.5.-Refrescar pantalla de captures	03/05/09	16/05/09
T10.- Posar estils i estructura a l'aplicació	17/05/09	02/06/09
T11.- Mostrar altres sniffers de la xarxa	03/06/09	08/06/09
T12.- Detalls finals	09/06/09	25/06/09

Figura 36: Evolució temporal

De fet el projecte estava previst ésser acabat el 19 d'abril i es va allargar fins el 24 de juny. A la planificació s'havia previst un projecte de 1160 hores i al final es van invertir un total de 1460 hores. Aquestes dades em porten a dir que el projecte es va allargar en un 30% del temps previst inicialment.

La majoria dels enderreriments van ésser deguts a dificultats per establir connexió entre els diferents elements del projecte i en entendre i instal·lar l'aplicació d'escriptori.

8.2 Anàlisi econòmic

Com he comentat anteriorment en aquest mateix apartat l'anàlisi econòmic final es veu afectat per enderreriments en el projecte. Tot seguit mostro la mateixa taula de l'apartat 4.4 de costs humans amb les dades reals posades en vermell (Figura 37).

PERFIL	HORAS	PREU(€/hora)	SUBTOTAL (€)
Jefe del projecte	60 (+15) =75	60	3600 (+900)=4500
Analista	300 (+40) =340	30	9000 (+1200)=10200
Programador	800 (+245)=1045	20	16000 (+4900)=20900
TOTAL	1160 (+300)=1460		20400 (+5075)=35600

Figura 37: Taula de costs del projecte

Cal recordar que en aquesta taula no es reflecteixen altres costs, com podria ser la preparació del projecte, la elaboració de la memòria, preparació de la presentació, entre altres.

Sumant tots els costs, la elaboració del projecte puja a 35.600 €. Els costs es poden veure desglossats a la següent taula (Figura 38)

TIPUS COST	QUANTITAT (€)
Hardware	196
Software	10
Recursos Humans	35600
TOTAL	**35906**

Figura 38: Taula de costs totals del projecte

Per tant, el preu projecte a augmentat 7.100 € després d'analitzar l'evolució real d'aquest.

9 Conclusions i treball futur

Els objectius els quals es van plantejar al començament del projecte han estat assolits, així com els requisits funcionals i els requisits no funcionals definits en el capítol 3 del projecte.

L'aplicació és robusta, eficient, sòlida i ràpida. Les tecnologies utilitzades, per tant, han estat encertades per tal de donar aquests valors a l'aplicació. Per exemple, el fet d'utilitzar dues màquines virtuals ha fet l'aplicació més ràpida, ja què d'aquesta manera es reparteix la carga de processament de les captures.

La utilització d'ajax ha fet l'aplicació molt més ràpida pel fet de no haver de recarregar la pàgina.

Pel què fa a la llibreria libpcap, part fonamental per a realitzar les captures, puc concloure que és una eina eficient i ràpida alhora de realitzar captures i comunicar-se amb el kernel de la màquina (Sistema Operatiu Linux). A més a més aquestes captures són més ràpides que no pas en sistemes windows ja que en aquests últims les captures i el filtratge es realitza dins la llibreria i no pas en el kernel de la màquina.

El temps de desenvolupament real de cada una de les tasques ha estat generalment més gran que no pas el calculat a l'anàlisi temporal previ al posterior desenvolupament. Aquest fet es deu a problemes i contratemps no prevists en aquesta etapa inicial.

Tot i que el temps de desenvolupament ha estat més del què s'esperava, la utilització del framework struts i de les llibreries JDOM i displayTag han ajudat a un ràpid desenvolupament de l'aplicació.

La metodologia escollida va ser la incremental ja que d'aquesta manera podia anar assolint metes petites que de mica en mica anaven donant cos a l'aplicació. De fet, va ser una bona opció ja que ha facilitat arribar a un objectiu sense pensar des de bon principi en la totalitat d'aquest, sinó fixar-se en petits aspectes al principi i anar afegint funcionalitats fins a arribar a l'objectiu principal, em va facilitar certes tasques.

Tot i que la finalització del projecte ha estat satisfactòria, a l'aplicació es podrien introduir canvis per tal de millorar-la. Es podrien noves tecnologies web i nous frameworks per a que fos més ordenada, segura i estable.

L'aplicació depen de moltes llibreries diferents. Per tant introduir “maven 2.0” podria ésser una bona solució per tenir un millor control de les llibreries que utilitza l'aplicació.

El sistema que consisteix en què l'ejb guardi les captures en un xml i que després recuperi l'aplicació des de Tomcat es podria millorar fent que l'ejb que es troba al servidor jboss llencés esdeveniments al Tomcat cada cop que hi hagués una captura. Així les captures no es guardarien, temporalment, en un XML i s'estalviarien recursos.

Amb l'objectiu de fer l'aplicació més estable i més sòlida i més mantenible es podria incorporar alguna de les llibreries junit ò mockito per tal de realitzar proves unitàries de l'aplicació.

Per altre banda una aplicació ha de segura. Ara per ara qualsevol persona pot accedir al sniffer i realitzar captures. Es podria, per exemple, incorporar el framework d' Spring i utilitzar l'springSecurity per tal de controlar-ne l'accés. També es podria utilitzar JOSSO el qual és un altre framework per donar seguretat de les aplicacions web. En els dos casos seria necessari la utilització d'una base de dades, per exemple Mysql.

Es podria utilitzar una base de dades i utilitzar el framework hibernate, per accedir-hi i per guardar-hi dades, per tal de guardar-hi certes captures i extreure'n gràfics o estadístiques amb alguna llibreria de generació de gràfics com podria ser JfreeChart. Fins hi tot extreure'n un PDF amb les estadístiques utilitzant jasperReports.

APENDIX 1: POSSIBLES ERRORS DE L'APLICACIÓ

En aquest apartat mostraré un llistat de possibles errors i les seves solucions que poden sorgir mentre instal·lem tota l'aplicació.

MICO

- error 1

/checking whether the C++ compiler (gcc) works... no

/configure: error: installation or configuration problem: C++ compiler cannot create executables.

- solució: Anar al repositori de llibreries del SO i instal·lar les llibreries de "gcc". En cas de no arreglar-se treure les posades i buscar alguna versió més actual.

- error 2

../include/mico/throw.h:111: error: 'strcmp' was not declared in this scope

- solució: Aquest error es deu a què la versió gcc que hi ha instal·lada no té alguna dependència. En aquest cas en concret es pot arreglar buscant l'arxiu throw.h i afegint (fer un include) la llibreria que conté la instrucció "strcmp" que és "stdio.h". En tot cas podria ésser millor canviar la versió de gcc o la de mico.

- error 3

/home/kimi/Desktop/mico/./idl/idl: error while loading shared libraries: libmicoir2.3.13.so: cannot open shared object file: No such file or directory

- solució: Com veiem mico no troba la ruta, per tant, em d'indicar-l'hi. Cal executar "export LD_LIBRARY_PATH=/(path)/mico/orb:$LD_LIBRARY_PATH" i "source /(path)/mico/admin/mico-steup.sh" .

- error 4

address.cc555 asertion failed

- solució: Aquest problema es deu a un error en la configuració de la màquina. Cal anar al fitxer "etc/hosts" , "etc/hosts.deny" i "etc/hosts.allow" i arreglar incongruències.

SERVER-BETA

- error 1

l/lib/libmico2.3.13.so: undefined reference to `SSL_CTX_use_certificate

- solució: El primer que cal fer és mirar si tenim la llibreria "libopenssl-devel" installada. En cas positiu executem "ldd (path)/libmico2.3.13.so" el resultat hauria de ésser :

linux-gate.so.1 => (0xffffe000)
libstdc++.so.6 => /usr/lib/libstdc++.so.6 (0xb7767000)
libm.so.6 => /lib/libm.so.6 (0xb7741000)
libgcc_s.so.1 => /lib/libgcc_s.so.1 (0xb7733000)
libc.so.6 => /lib/libc.so.6 (0xb75f0000)
/lib/ld-linux.so.2 (0xb7f91000)

Si no obtenim aquest resultat cal tornar a instal·lar MICO. En cas positiu hem de mirar el fitxere Makefile.am i mirar si tenim incorporat "-lssl" a la compilació "(dimoni)_LDFLAGS".

CONNEXIÓ TOMCAT JBOSS

- error 1

java.lang.ClassCastException: $Proxy2 cannot be cast to main.java.com.upc.sniffer.service.CaptureBean

- solució: Pot ésser degut a una mala ubicació del jar de l'ejb. L'ejb ha d'estar en el classpath de l'aplicació web. Amb l'eclipse s'ha de configurar el Build Path

afegint el jar.

- error 2

javax.naming.NamingException: Could not dereference object [Root exception is java.lang.reflect.UndeclaredThrowableException]
at org.jnp.interfaces.NamingContext.getObjectInstanceWrapFailure (NamingContext.java:1463)
at org.jnp.interfaces.NamingContext.lookup(NamingContext.java:809)
at org.jnp.interfaces.NamingContext.lookup(NamingContext.java:673)
at javax.naming.InitialContext.lookup(InitialContext.java:392)
at main.java.com.uni.sniffer.actions.PresentCaptureAction$StartCap.run (PresentCaptureAction.java:126)
Caused by: java.lang.reflect.UndeclaredThrowableException
at $Proxy0.createProxyBusiness(Unknown Source)
at org.jboss.ejb3.proxy.objectfactory.session.SesionProxyObjectFactory. createProxy (SessionProxyObjectFactory.java:129)
at org.jboss.ejb3.proxy.objectfactory.session.stateless. StatelessSessionProxyObjectFactory.getProxy (StatelessSessionProxyObjectFactory.java:79)
at org.jboss.ejb3.proxy.objectfactory.ProxyObjectFactory. getObjectInstance(ProxyObjectFactory.java:156)
at javax.naming.spi.NamingManager.getObjectInstance (NamingManager.java:304)
at org.jnp.interfaces.NamingContext.getObjectInstance (NamingContext.java:1438)
at org.jnp.interfaces.NamingContext.getObjectInstanceWrapFailure (NamingContext.java:1455)
... 4 more
Caused by: java.lang.ClassNotFoundException: ejb.CaptureBeanRemote
at org.jboss.remoting.serialization.ClassLoaderUtility. loadClass(ClassLoaderUtility.java:103)
at org.jboss.remoting.loading.RemotingClassLoader. loadClass(RemotingClassLoader.java:86)
at java.lang.ClassLoader.loadClassInternal(ClassLoader.java:319)
at java.lang.Class.forName0(Native Method)
at java.lang.Class.forName(Class.java:247)
at org.jboss.remoting.loading.ObjectInputStreamWithClassLoader. resolveProxyClass(ObjectInputStreamWithClassLoader.java:250)
at java.io.ObjectInputStream.readProxyDesc (ObjectInputStream.java:1531)
at java.io.ObjectInputStream.readClassDesc(ObjectInputStream.java:1493)
at java.io.ObjectInputStream.readOrdinaryObject

(ObjectInputStream.java:1732)

- solució: Aquest error es pot produir per més d'un motiu. Primer cal comprovar que tant JBOSS com Tomcat utilitzen el mateix Runtime Environment (JDK) ja que és necessari per al bon funcionament de l'aplicació. Un cop verificat i si segueix l'error comprovar si és degut als permisos del JRE. Cal afegir permisos al java.policy del JRE. Per esbrinar si és un error de permisos cal permetre totes les operacions afegint a l'arxiu java.policy afegint "permission java.security.AllPermission;".

- error 3

INFO [STDOUT] [WorkerThread#0[127.0.0.1:28169]] [ERROR] (orb):
IOException while parsing XML File
"resource:/org/openorb/config/OpenORB.xml"
java.io.FileNotFoundException: Could not locate resource:
/org/openorb/config/OpenORB.xml
at org.jboss.net.protocol.resource.ResourceURLConnection.
makeDelegateUrl(ResourceURLConnection.java:74)
at org.jboss.net.protocol.DelegatingURLConnection.
<init>(DelegatingURLConnection.java:54)
at org.jboss.net.protocol.resource.ResourceURLConnection.
<init>(ResourceURLConnection.java:49)

- solució: Aquest error es deu a què JBOSS no troba el OpenORB.xml. JBOSS no busca dins del lib del JRE sinó dins de l'ejb. Per tant vaig haver de crear una carpeta org.openorb.config i a dins posar els fitxers OpenORB.xml i default.xml.

- error 4

Exception java.lang.NoSuchField Error: TRACE
at org.jboss.logging.log4jLoggerPlugin.isTraceEnabled(Log4jLoggerPlugin....)
at javax.naming.InitialContext.lookup

- solució: Aquest error és degut a la versió de log4j. Cal actualitzar la versió.

- error 5

java.lang.ClassNotFoundException: org.omg.CosEventComm

- solució: Si tomcat no troba les classes del jre assegurar-se de que es compila i s'executa amb el mateix JRE que jboss. Posar variable entorn "export JAVA_HOME=/*" i posar-la també a catalaina.sh.

- error 6

java.net.UnknownHostException: dhcppc3: dhcppc3

- solució: Aquest error vol dir que hi ha errors de configuració del nom de la màquina dins el fitxer "/etc/hosts"

- error 7

java.lang.Exception: Port 8083 already in use.

- solució: Aquest error pot ésser degut a una mala configuració del apache i de jboss els quals estan utilitzant simultàniament el port 8083. Pot ésser degut a que el servidor que utilitza aquest port ja ha estat llençat; caldrà parar-l'ho i tornar-l'ho a llençar.

- error 8

[com.arjuna.ats.internal.arjuna.utils.SocketProcessId_2] - SocketProcessId.getpid could not get unique port.

- solució: Aquest error és produït majoritàriament per errors en la configuració de l'ip de la màquina dins el fitxer "/etc/hosts". Mirar l'ip de la màquina executant "./ifconfig" en el directori "/sbin/" i comprovar que sigui el que tenim a "/etc/hosts".

- error 9

javax.naming.NameNotFoundException: Name ejb is not bound in this Context
at org.apache.naming.NamingContext.lookup(NamingContext.java:768)
at org.apache.naming.NamingContext.lookup(NamingContext.java:151)
at org.apache.naming.SelectorContext.lookup(SelectorContext.java:136)
at javax.naming.InitialContext.lookup(InitialContext.java:392)
at main.java.com.uni.sniffer.actions.PresentCaptureAction$StartCap.run (PresentCaptureAction.java:107)

- solució: Aquest error es degut a que quan tomcat fa el lookup per connectar amb l'ejb no el troba. Cal assegurar-se de que jboss ha arrencat correctament i mirar de no tenir el jar de l'ejb duplicat a l'aplicació que corre sota tomcat.

CONNEXIÓ JBOSS SERVER-BETA

- error 1

snifer entry: null
main.java.com.uni.sniffer.exceptions.DaemonException: Sniffer is not in network

- solució: Aquest error es deu a què segurament el dimoni del servidor de captures no ha estat llençat.

- error 2

INFO: addCapture Daemon does not exist-->
org.omg.CORBA.TRANSIENT: vmcid: 0x0 minor code: 2 completed: No
at org.openorb.orb.core.Delegate$RequestState.findNextBinding (Delegate.java:2258)
at org.openorb.orb.core.Delegate$RequestState.failoverFatal (Delegate.java:2337)
at org.openorb.orb.core.Delegate$RequestState.receiveSystemException (Delegate.java:2395)
at org.openorb.orb.core.Delegate$RequestState.access$000 (Delegate.java:1583)
at org.openorb.orb.core.Delegate.request(Delegate.java:957)
at org.openorb.orb.core.Delegate.is_a(Delegate.java:611)
at org.omg.CORBA.portable.ObjectImpl._is_a(ObjectImpl.java:112)
at daemons.SnifferHelper.narrow(SnifferHelper.java:103)
atmain.java.com.uni.sniffer.service.addCapture.addCapture (addCapture.java:77)
at main.java.com.uni.sniffer.actions.SolicitaCaptureAction.execute (SolicitaCaptureAction.java:61)
at org.apache.struts.chain.commands.servlet.ExecuteAction.execute (ExecuteAction.java:58)

-solució: Aquest error es deu a què el dimoni sniffer no s'ha llençat correctament.

BIBLIOGRAFIA

A continuació es detalla la bibliografia més important. Quasi tot el material consultat fa referència a pàgines web ja que internet és una font d'informació molt important.

- T4-Middleware-RMI-CORBA-parte2.pdf . [En línia]. Disponible a: www.dia.eui.upm.es .
- 9_services.pdf. [En línia]. Disponible a: <http://www.it.uc3m.es/spickin/docencia/sisinf/slides/9_services.pdF>.
- CORBA.ppt. [En línia]. Disponible a: <http://profile.iiita.ac.in/vksonker_is04/other/CORBA.ppt>.
- Corba-and-java.ppt. [En línia] Disponible a: <http://www.purpletech.com/talks/CORBA/corba-and-java.ppt>.
- CORBASilarri.pdf. [En línia] Disponible a: <http://webdiis.unizar.es/~jmerse/IS-2/ObjetosDistribuidos/CORBASilarri.pdf>.
- IntroduccionSistemasDistribuidos_SE.pdf. [En línia]. Disponible a: <http://observatoridelacapacitacion.stps.gob.mx>.
- EJB30_3_.pdf. [En línia]. Disponible a: <http://dis.um.es>.
- JbossPortalreferenceGuide.pdf. [En línia]. Disponible a: <http://docs.jboss.org/jbportal/v2.6.2/>.
- Introducao_ao_MICO.pdf. [En línia]. Disponible a: <http://www.dei.isep.ipp.pt/~alex/publico/sd/Introducao_ao_MICO.pdf>.
- Mapeo-IDL-a-C++.ppt. [En línia]. Disponible a: <http://www.face.ubiobio.cl/~prodrigu/mad/Mapeo-IDL-a-C++.ppt>.
- Pcap.pdf. [En línia]. Disponible a: www.e-ghost.deusto.es/docs/2005/conferencias/pcap.pdf
- Utilitats dels sniffers. [En línia]. Disponible a: http://www.terra.es/tecnologia/articulo/html/tec7425.htm

- 6.pdf. [En línia]. Disponible a: www.deltapublicaciones.com/docus/6.pdf

- introduccion_ajax.pdf. [En línia]. Disponible a: <http://www.librosweb.es/ajax/pdf/introduccion_ajax.pdf>.

- Instalacion-y-Configuracion-del-JBOSS. [En línia]. Disponible a: <http://www.scribd.com/doc/3356121/Instalacion-y-Configuracion-del-JBOSS>.

- WIKIPEDIA. [En línia]. Disponible a: <http://es.wikipedia.org>.

- PÀGINA WEB DE TOMCAT. [En línia]. Disponible a: <http://tomcat.apache.org/>.

- PÀGINA WEB DE CORBA. [En línia]. Disponible a: <http://www.corba.org>.

- PÀGINA WEB DE MICO. [En línia]. Disponible a: <http://www.mico.org>.

- PÀGINA WEB D' STRUTS APACHE. [En línia]. Disponible a: <http://struts.apache.org>.

- PÀGINA WEB DE JDOM. [En línia]. Disponible a: <http://www.jdom.org>.

- PÀGINA WEB DE DISPLAYTAG. [En línia]. Disponible a: <http://displaytag.sourceforge.net>.

- PÀGINA WEB DE JBOSS. [En línia]. Disponible a: <http://www.jboss.org>.